TREBALL DE FI DE CARRERA

TÍTOL: Processament a nivell de bit en capes baixes de WLAN sobre DSP

AUTOR: Eduardo Hervás Rosa

DIRECTOR: Xavier Revés Ballesté

DATA: 21 de Juliol de 2006

Títol: Processament a nivell de bit en capes baixes de WLAN sobre DSP

Autor: Eduardo Hervás Rosa

Director: Xavier Revés Ballesté

Data: 21 de Juliol de 2006

Resum

L'objectiu d'aquest TFC és realitzar els blocs que constitueixen les etapes de processament de bits que estan definides dins la capa física de l'estàndard IEEE 802.11a (vàlides per la resta de normes de la família 802.11 que utilitzen OFDM com a tècnica de modulació).

Per aconseguir l'objectiu s'han realitzat una sèrie de mòduls programats en C fent ús d'una eina d'abstracció de la plataforma sota la que s'executa el programa, Linux o bé una DSP, anomenada Communications Manager (CM).

Els mòduls es poden avaluar individualment i, un cop comprovat el seu correcte funcionament, es poden integrar amb altres blocs també desenvolupats sobre CM mitjançant interfícies gestionades pel propi CM. El procés d'integració és poc costós ja que els mecanismes de connexió dels blocs estan definits a priori.

Així doncs, les tasques realitzades en la realització d'aquest TFC són les següents:

1) Analitzar el funcionament de la capa física de l'estàndard IEEE 802.11a.

2) Realitzar el disseny de l'estructura del software de la subcapa PLCP del transmissor i del receptor, l'encarregada del processament dels bits.

3) Adaptar el disseny del software per a que sigui adient a les regles imposades per CM.

4) Realitzar els programes corresponents als diferents blocs o mòduls que s'encarreguen del processament de bits.

5) Validar-ne el correcte funcionament en el context de CM a partir d'un banc de proves per als diferents modes de funcionament.

Title: Processing at bit level in low layers of WLAN through DSP

Author: Eduardo Hervás Rosa

Director: Xavier Revés Ballesté

Date: July, 21th 2006

Overview

The main scope of this TFC is making the blocks what form bit processing steps defined in the Physical Layer of the standard IEEE 802.11a (valid for the rest of norms for 802.11 family using OFDM as modulation technique).

To reach these objectives have been programmed some modules in C language using a plattform abastraction tool over program is executed, Linux or a DSP, called Communications Manager (CM).

Modules can be evaluated individually and once verified their fine behaviour they can be integrated with other blocks developped over CM through interfaces managed by the same CM. Integration process is not complicated because connexion mechanisms were defined before.

Therefore, the tasks performed for this TFC are:

1) To analyse the physcial layer behavior of the standard IEEE 802.11a.

2) To design the software structure of PLCP sublayer for transmitter and receiver, that performs bit processing.

3) Adapting software design to meet CM rules.

4) Programming blocks or modules performing bit processing.

5) Check fine behaviour over CM context through a testbed for the different configurations.

ÍNDEX

INTRODUCCIÓ

La possibilitat de connectar-se a una xarxa sense cables és a dia d'avui una realitat a l'abast de tothom. L'actual tendència en l'àrea de comunicacions s'orienta a la utilització de sistemes portàtils reconfigurables amb alta velocitat de connexió i elevat nivell de seguretat.

La tecnologia OFDM sembla ser una de les tècniques amb més futur degut a l'alta velocitat que es pot aconseguir a més de l'elevada eficiència espectral.

L'objectiu d'aquest TFC és el de realitzar els blocs que constitueixen les etapes de processament de bits que estan definides dins la capa física de l'estàndard IEEE 802.11a (vàlides per la resta de normes de la família 802.11 que utilitzen OFDM com a tècnica de modulació). Aquestes etapes es programaran en llenguatge C adaptable a diferents plataformes, entre elles les basades en processadors digitals de senyal (DSP) i els PCs amb sistema operatiu Linux. Per tal de fer que els blocs desenvolupats es puguin utilitzar sobre les esmentades plataformes, es farà ús d'una eina d'abstracció de la plataforma prèviament desenvolupada i coneguda com a Communications Manager (CM). Els blocs, un cop provats sobre una plataforma genèrica com un PC amb Linux, es poden utilitzar sobre les plataformes DSP sense cap problema de compatibilitat si s'han respectat les regles imposades per l'eina d'abstracció.

Els blocs es poden avaluar individualment i, un cop comprovat el seu correcte funcionament, es poden integrar amb altres blocs també desenvolupats sobre CM mitjançant interfícies gestionades pel propi CM. El procés d'integració és poc costós ja que els mecanismes de connexió dels blocs estan definits a priori. Així doncs, les tasques planificades per a la realització d'aquest TFC són les següents: 1) analitzar el funcionament de la capa física de l'estàndard IEEE 802.11, 2) realitzar el disseny de l'estructura del software de la subcapa PLCP del transmissor i del receptor, l' encarregada del processament dels bits, 3) adaptar el disseny del software per a que sigui adient a les regles imposades per CM, 4) realitzar els programes corresponents als diferents blocs o mòduls que realitzen el processament i 5) validar-ne el correcte funcionament en el context de CM a partir d'un banc de proves per als diferents modes de funcionament.

L'estructura a nivell de capítols d'aquesta memòria és la següent:

- El capítol 1 conté una introducció a les xarxes sense fils (WLAN), els diferents estàndards de la norma 802.11 i finalment avantatges i inconvenients d'aquesta tipologia de xarxa.

- El capítol 2 presenta la capa física -PHY- i fa una explicació de la tècnica de modulació OFDM.

- El capítol 3 especifica els diferents camps que formen un paquet complet de dades i les funcions que s'han de realitzar per poder emetre els bits en cas d'emissió i per poder-los rebre en recepció.

- El capítol 4 introdueix l'aplicació Comunicacions Manager sota la que s'executa l'aplicació i fa una explicació acurada dels mòduls que la conformen.

- El capítol 5 recull les conclusions obtingudes d'aquest TFC.

CAPÍTOL 1. WLAN

1.1 Introducció

Una xarxa d'àrea local sense fils (en anglès Wireless Local Area Network, WLAN), pot definir-se com una xarxa de curt abast (aproximadament uns 80 m.) que té com mitjà de transmissió l'aire i aconsegueix velocitats de transferència de dades relativament grans (>1Mbps) amb una baixa taxa d'errors. Aquestes xarxes utilitzen ones electromagnètiques a freqüències de microona per connectar els diferents equips o terminals associats a la xarxa.

La tecnologia sense fils permet crear sistemes de comunicacions flexibles que poden implementar-se com una extensió o directament com una alternativa a una xarxa cablejada. Això fa que l'usuari pugui moure's dins de l'àrea de cobertura de la xarxa sense perdre la connexió. S'aconsegueix utilitzant punts d'accés (AP) per radiar les ones electromagnètiques.

A la figura **(Fig. 1.1)** s'observen els elements bàsics que conformen una WLAN.

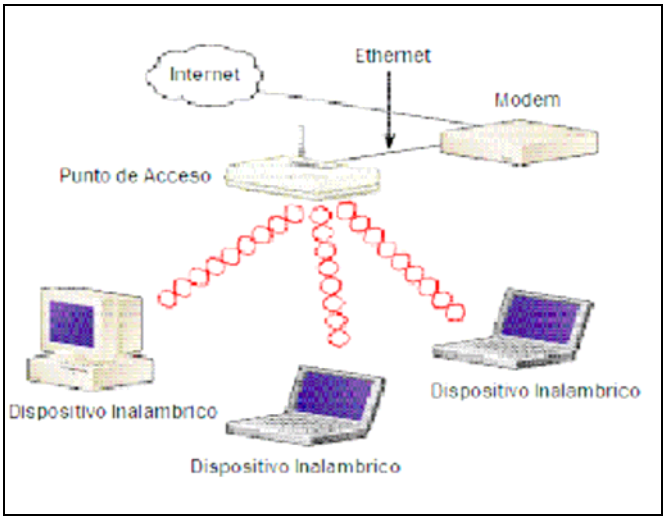


Fig. 1.1 Elements bàsics d'una WLAN

L'origen de les WLAN data de l'any 1979 amb la publicació dels resultats d'un experiment desenvolupat per IBM consistent en utilitzar enllaços infrarojos (IR) per crear una xarxa local dins una fàbrica. Les investigacions van continuar endavant tant amb IR com amb microones.

El maig del 1985 la Comissió Federal de Comunicacions dels EEUU (FFCC) va assignar les bandes Industrial, Científica i Mèdica (en anglès Industrial, Scientific i Medical ISM) amb les freqüències 902-928 MHz, 2.400-2.4835 GHz, 5.725-5.850 GHZ respectivament a les WLAN basades en la modulació d'Spread Spectrum (tècnica ideal per aquest tipus de sistemes ja que genera molt poca interferència i és força immune al soroll) [1].

L'Institut d'enginyers elèctrics i electrònics (en anglès Institute of Electrical and Electronic Engineers –IEEE-) va interessar-se molt per les WLAN i el 1989 van formar el comitè IEEE 802.11.

Poc a poc, les xarxes locals abandonaven els laboratoris per introduir-se al mercat comercial. Tot i que es va treballar molt dur, no fou fins el maig del 1991 quan es van publicar diferents treballs referents a les WLAN amb velocitats de fins 1 Mbps.

1.2 Estàndards

Els estàndards constitueixen la pedra angular en el desenvolupament de les xarxes sense fils, ells mateixos han estat fonamentals pel creixement del sector en els últims anys. La seva importància radica en que defineixen el conjunt de característiques tècniques que han de seguir els fabricants en el moment de realitzar els seus productes, permetent així que dispositius de marques diferents siguin compatibles entre sí i compleixen amb els requisits mínims.

Per exemple, els equips que compleixen amb els estàndards 802.11b i 802.11g incorporen el logo de la figura **(Fig. 1.2)** indicant la compatibilitat entre equips.

Fig. 1.2 Logo compatibilitat WiFi

Existeixen vàries organitzacions que es dediquen a establir els paràmetres tècnics d'un estàndard, les més importants són la IEEE (Nord-americana) i la ETSI (Europea) [2].

Cada estàndard defineix el conjunt de característiques de la Capa Física –PHY- (capa 1) i de la Capa d'Enllaç de Dades o –MAC- (capa 2), **(Fig. 1.3)** totes dues del model de referència OSI.
A continuació es presenta una breu descripció dels principals estàndards de les xarxes sense fils.

Fig. 1.3 Elements de la Capa OSI

1.2.1 IEEE 802.11

El 1997 la IEEE publica l'estàndard 802.11, el primer de xarxes sense fils que té acceptació a nivell internacional. Aquest proposa tres formes diferents de transmissió per la capa física: la primera especifica la transmissió mitjançant infrarojos (tècnica que mai es va implementar comercialment) i les altres dues proposaven una modulació d'espectre eixamplat per seqüència directa (en anglès Direct Sequence Spread Spectrum –DSSS-), o bé, d'espectre eixamplat per salt de freqüència (en anglès Frequency Hopping Spread Spectrum – FHSS-). Totes dues treballant a la banda ISM de 2.4 GHz [1].

A la modulació DSSS es genera un patró de bits redundants (senyal de xip) per cada un dels bits que composen el senyal. Quant més gran sigui el senyal, més gran serà la resistència del senyal a les interferències. L'estàndard 802.11 recomana un tamany d'11 bits, però l'òptim és 100. En recepció és necessari realitzar el procés invers per obtenir la informació original.

La seqüència de bits utilitzada per modular es coneix com seqüència de Barrer (també anomenat codi de dispersió o PseudoNoise). És una seqüència ràpida dissenyada per que aparegui aproximadament la mateixa quantitat de 1's que de 0's. Un exemple d'aquesta seqüència és el següent: +1-1+1+1-1+1+1+1-1-1-1-1-1. Només els receptors als quals l'emissor hagi enviat prèviament la seqüència podran recompondre el senyal. A més, al substituir cada bit de dades a transmetre per una seqüència d'11 bits equivalent, tot i que part del senyal es vegi afectat per interferències, el receptor encara pot reconstruir fàcilment la informació a partir del senyal rebut. Per un receptor aliè, DSSS apareix com un senyal de soroll amb un ample de banda de baix poder que és ignorat per la resta de receptors.

La majoria dels fabricants de productes WLAN han adoptat la tecnologia DSSS després de considerar els beneficis vs costos i rendiment que s'obtenen.

FHSS utilitza un senyal portador que canvia de freqüència amb un patró que és conegut per transmissor i receptor.

Correctament sincronitzada, la xarxa efectua aquest canvi per mantenir un únic canal lògic d'operació. FHSS commuta entre una banda estreta i una altra aproximadament 10 vegades per segon.

Les velocitats màximes que s'aconsegueixen a la capa física amb qualsevol de les dues tècniques és de 1 a 2 Mbps depenent del nivell de senyal, tenint en compte que la velocitat efectiva de transmissió de dades és menor degut a la utilització de capçaleres a la capa MAC [3].

1.2.2 IEEE 802.11b

Revisions de l'estàndard anterior (802.11) van provocar la recerca de millores en la velocitat de transmissió. Per aconseguir-ho es crea el "Grup de Treball b" el qual tenia com a meta augmentar la velocitat mitjançant l'ús de tecnologies DSSS més sofisticades [4].

Els resultats arriben el 1999 quan publiquen l'estàndard 802.11b i poc després aquest esdevé un èxit actualment present.

La capa física de l'estàndard especifica l'ús de la modulació DSSS a la banda de 2.4 GHz amb un ample de banda total de 83.5 MHz dividits en 11 canals de 22 MHz, els quals es solapen entre sí **(Fig. 1.4)**, existint només 3 canals independents (canals 1, 6 i 11).

Les velocitats de transmissió són 1, 2, 5.5 i 11 Mbps depenent del nivell de potència del senyal.

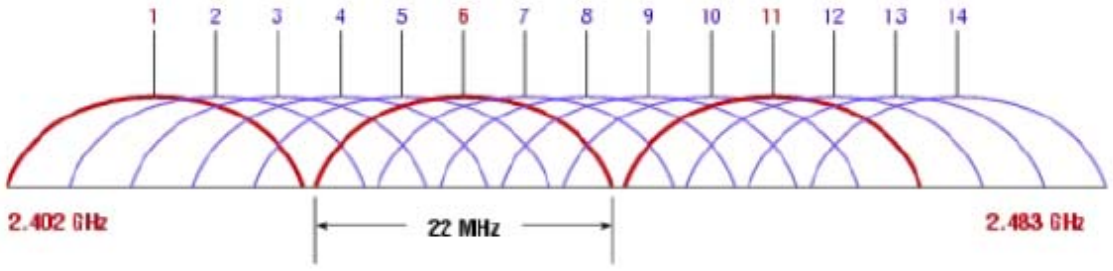


Fig. 1.4 Canals a la banda de 2.4 GHz

1.2.3 IEEE 802.11a

L'any 1999 la IEEE publica l'estàndard 802.11a, també conegut com WI-FI 5 (en anglès Wireless Fidelity 5) a la banda dels 5 GHz (banda U-NII) amb l'objectiu d'obtenir major ample de banda i evitar la banda de 2.4 GHz que ja es trobava força saturada. Aquest estàndard fou el primer que utilitzà la modulació OFDM (Ortoghonal Frequency Division Multiplexing) la qual serà presentada més endavant en aquest mateix document [3]. A Europa aquest estàndard, el 802.11a, no ha tingut massa acceptació perquè fins fa relativament poc, l'any 2003, aquesta banda de freqüències, els 5 GHz, no era lliure, fet que va promoure la proliferació del 802.11b [4].

La banda dels 5 GHz disposa de 300 MHz d'ample de banda (en lloc dels 80 MHz de la banda ISM) i és molt més immune a interferències ja que els dispositius que treballen en aquesta banda són molts menys.

La banda U-NII es va dividir en 12 canals independents de 20 MHz cadascú per aprofitar al màxim l'ample de banda disponible amb l'ajut de la OFDM.

Les velocitats de transmissió són 6, 9, 12, 18, 24, 36, 48 i 54 Mbps.

1.2.4 IEEE 802.11g

Aquest estàndard apareix l'any 2003. El principal objectiu de disseny era augmentar la velocitat màxima aconseguida (11 Mbps) a la banda ISM per part dels estàndards anteriors mantenint compatibilitat amb el seu predecessor, el 802.11b, ja que aquest gaudia de molt bona salut a nivell comercial.

La capa física del 802.11g utilitza dues modulacions, la OFDM i la DSSS (aquesta última per compatibilitat amb el 802.11b) a la banda ISM de 2.4 GHz. El tamany dels canals i la seva ubicació és la mateixa que al 802.11b.

Les velocitats de transmissió varien entre 1, 2, 5.5, 6, 9, 11, 12, 18, 24, 36, 48 i 54 Mbps.

1.2.5 Sumari

A la taula **(Taula 1.1)** es pot veure un resum de les principals característiques dels estàndards presentats i d'altres (vigents actualment) però que es surten de l'interès d'aquest treball per ser evolucions dels estàndards abans citats sense modificacions a la capa física o bé, perquè la modulació utilitzada no és propòsit d'estudi en aquest cas.

Taula 1.1. Resum d'estàndards i especificacions

Estàndard	802.11a	802.11b	802.11g	802.11h	HiperLAN/2	*Bluetooth*
Organisme	IEEE	IEEE	IEEE	IEEE	ETSI	Bluetooth/SIG
Finalització	2002	1999	2003	2003	2003	2002
Banda de freqüència	5GHz	2,4GHz	2,4GHz	5GHz	5GHz	2,4GHz
Velocitat màxima	54Mbits/s	11Mbits/s	54Mbits/s	54Mbits/s	54Mbits/s	0.721Mbits/s
Tipus de modulació	OFDM	DSSS	DSSS/OFDM	OFDM	OFDM	FHSS

1.3 Avantatges de les WLAN

Les WLAN són recomanades en les següents situacions:

- Implementació de xarxes d'àrea local (LAN) en llocs de difícil accés i en general en entorns on la solució cablejada és inviable.
- Possibilitat de reconfiguració de la topologia de la xarxa sense afegir costos adicionals. (Situació típica en entorns variants que requereixen una estructura de xarxa flexible davant aquests canvis).
- Xarxes locals per situacions d'emergència o congestió de la xarxa cablejada.
- Accés a la informació mentre l'usuari es troba en moviment. (Típic en hospitals, fàbriques, empreses,…)
- Generació de grups de treball eventuals i reunions ad-hoc. En aquests casos la instal·lació temporal d'una xarxa cablejada és inviable.
- Interconnexió de diferents dispositius i màquines en ambients industrials.
- Interconnexions de xarxes d'àrea local que es troben en emplaçaments diferents.

1.4 Inconvenients de les WLAN

A continuació es presenten els principals inconvenients de les WLAN.

- Susceptibilitat a interferències produïdes per dispositius que funcionen en la mateixa banda que aquests (telèfons, dispositius bluetooth, forns microones,…)
- La seguretat és menor que la d'una xarxa cablejada. La informació és irradiada i viatja en clar si no està encriptada.
- La seguretat WEP ha estat clarament demostrada insuficient.

- Diferents usuaris comparteixen el mateix canal, en conseqüència la velocitat de transmissió baixa.
- La velocitat de transmissió és com a màxim 54 Mbps, mentre que a una xarxa cablejada es pot arribar a velocitats d'1Gbps.
- La capçalera de les trames és més gran que la de xarxes cablejades.

CAPÍTOL 2. CAPA FÍSICA (PHY)

2.1 La capa física (PHY)

La Capa Física estableix les especificacions que permeten transformar els paquets de dades procedents de la capa MAC en senyals elèctrics analògics apropiats per ser transmesos al medi, en aquest cas l'aire. També aquesta capa s'encarrega de realitzar el procés invers quan es rep informació, és a dir, captar senyals presents a l'aire i transformar-los en paquets de dades binaris que seran entregats a la capa MAC.

Quan un terminal realitza una transmissió, la informació arriba a la capa física des de la capa MAC en forma de paquets, també coneguts com Protocol Data Unit –MPDU- els quals estan composats per bits. Aquests han de ser codificats en senyals elèctrics que posteriorment seran modulats i transmesos al medi mitjançant antenes. Si pel contrari el terminal es troba rebent dades, aquest procés és l'invers, és a dir, l'antena capta el senyal que es troba en el medi i el demodula, amb això s'obté un tren de bits que es descodifica i es transforma en un paquet apte per ser entregat a la capa MAC.

La modulació d'un senyal es realitza en dues etapes, una en bandabase i l'altra en passabanda. A la primera es realitza la codificació de línia, la qual consisteix en convertir els paquets finals a transmetre –PPDU- en formes d'ona binàries utilitzant algun codi de línia tipus Manchester, NRZ, Bipolar, etc. A la segona etapa es realitza una modulació passabanda, consistent en mapar la forma d'ona binària en una portadora d'alta freqüència, i aconseguir així elevar la freqüència del senyal resultant per poder-lo transmetre fent us d'antenes de tamanys físicament raonables. S'utilitzen varis tipus de modulació digital per realitzar aquest procés (QAM, BPSK, etc.).

A l'estàndard 802.11a (que defineix la capa física de tots els estàndards 802.11 que utilitzen la OFDM com a tipus de modulació) la capa física s'ha dividit en dues subcapes: la subcapa Physical Medium Dependent (PMD) i la subcapa Physical Layer Convergence Procedure (PLCP). La figura **(Fig. 2.1)** mostra gràficament aquesta divisió.

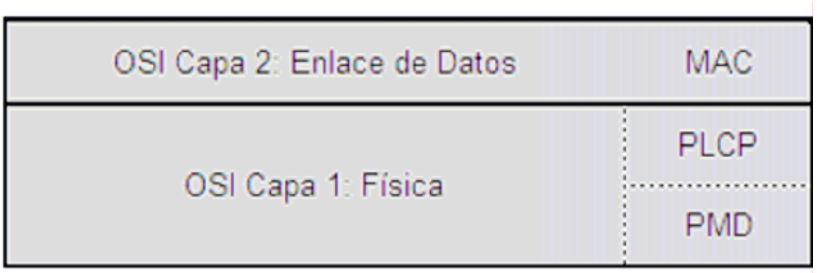


Fig.2.1 Divisió de la capa física (PHY)

- **PMD:** en aquesta subcapa es defineixen les característiques físiques i els mètodes per enviar i rebre les dades mitjançant el medi Wireless. Es fa càrrec dels següents paràmetres: sincronització, detecció del medi, tipus de senyal, freqüència d'operació, amplitud de la senyal, esquema de modulació i tota la resta de característiques de tipus físic.

- **PLCP:** aquesta subcapa estableix la forma en que els MPDU procedents de la capa MAC seran mapats en la capa física per convertir-los en paquets adequats per ser entregats al medi, és a dir, tradueix el paquet MAC en un paquet físic, i al contrari, de físic a MAC en cas de recepció.

La capa física d'un medi cablejat i la d'un Wireless són força diferents, a continuació alguna d'aquestes diferències respecte el cable:

- Les transmissions no estan protegides de senyals externs.
- El mitjà és menys confiable, la seva funció de transferència varia.
- La topologia de la xarxa és dinàmica.
- L'atenuació és elevada i varia d'acord la ubicació de la xarxa.
- No es pot garantir que una estació detecti les transmissions de la resta, tot i que aquestes pertanyin al mateix domini de col·lisió.
- L'ample de banda disponible és molt menor.

2.2 OFDM

La OFDM és una tècnica de transmissió que data de l'any 1970. En aquell moment la utilització d'aquesta era poc viable degut a que requereix grans capacitats de processament de senyal i la tecnologia disponible per la seva implementació era massa costosa. Actualment aplicacions com DSL, WLAN i TV Broadcasting han posat la seva mirada en OFDM gràcies a la seva eficiència espectral i la seva resistència contra els esvaïments per multipath. L'alta capacitat de processament

dels circuits integrats actuals i el baix cost fan possible la implementació d'aquesta tècnica de forma eficient i econòmica [5][6][7].

La figura **(Fig. 2.2)** proporciona una idea de la quantitat de portadores que s'envien en un determinat ample de banda utilitzant FDM i OFDM, és interessant observar com usant OFDM la quantitat augmenta considerablement.

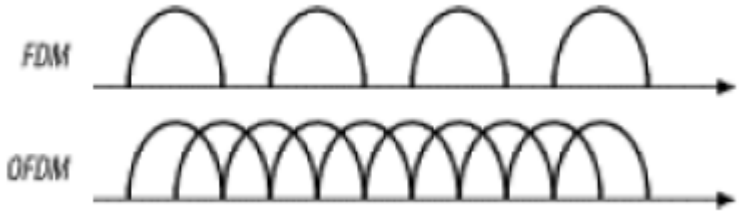


Fig. 2.2 Comparació espectral entre FDM i OFDM

En termes generals, la OFDM es basa en dividir el canal de comunicacions en el domini de la freqüència en varis canals més petits, en cadascun dels quals es transmet una subportadora. Cada una de les N subportadores que es transmet en els N subcanals deuen ser ortogonals entre sí, d'aquesta manera es permet el solapament de les mateixes sense que això provoqui interferències.

La informació que s'envia és multiplexada en les subportadores i es transmet llavors de forma paral·lela, per tant, ara en lloc d'enviar una portadora que utilitzi tot l'ample de banda disponible, s'envien vàries subportadores amb un ample de banda N vegades menor. Aquesta tècnica permet un millor aprofitament de l'ample de banda del canal gràcies a que les subportadores es poden solapar, evitant d'aquesta manera bandes de guarda.

2.2.1 Descripció formal de la OFDM

OFDM és un esquema de modulació en el qual els símbols són transmesos en paral·lel utilitzant un número considerable de subportadores ortogonals. Un bloc de N símbols es transmet en sèrie en T_s segons cadascú i es converteix en un bloc de N símbols en paral·lel que es transmeten en $T = n \cdot T_s$ segons cadascú **(Fig. 2.3)**.

Els símbols tenen llavors una duració N vegades major permetent així reduir la interferència intersimbòlica (en anglès Inter Symbol Interference –ISI-), això es deu a que al tenir símbols més grans, el percentatge d'afectació d'un símbol per un adjacent és menor. A cadascú dels símbols el correspon modular una de les N

subportadores, és a dir, en cas de tenir N símbols calen N subportadores, cada una de les quals deu estar separada 1/T Hz (per mantenir l'ortogonalitat).

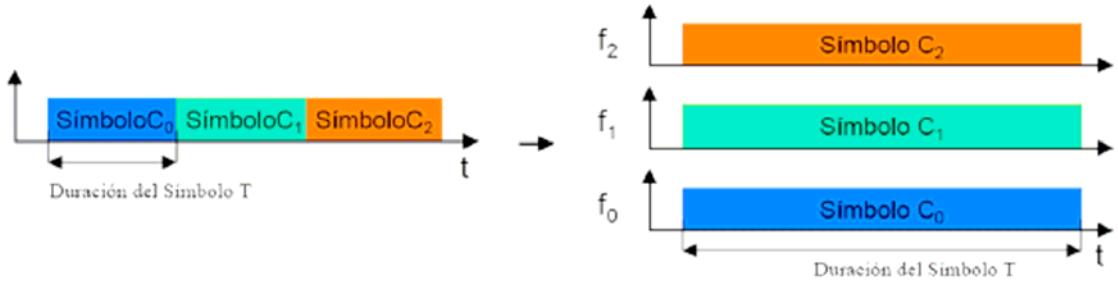


Fig. 2.3 Transformació dels símbols

El senyal OFDM en bandabase es defineix de la següent manera **(2.1)** :

$$v(t) = \sum_{k=0}^{N-1} I_k \cdot e^{j2\pi k \cdot t/T} si 0 \le t \le T \qquad \textbf{(2.1)}$$

On:

I_k: Símbol complex. Aquest conté la informació.
N: Número de subportadores.
k: Índex de la subportadora.
T: Temps del bloc OFDM.
1/T: Freqüència de separació entre les subportadores per tal que siguin ortogonals.
v(t): Senyal OFDM.

Per evitar ISI, producte del multipath, s'afegeix al començament del símbol un interval de guarda (en anglès Guard Interval –GI-), el qual és una còpia de la part final del símbol. Aquest procediment es coneix amb el nom de prefix cíclic. La duració mínima del GI per poder eliminar la ISI ha de ser major que l'endarreriment introduït pel canal. La figura **(Fig. 2.4)** mostra la col·locació del GI. A continuació es presenta el senyal que s'obté després d'incorporar el prefix cíclic **(2.2)**.

$$v(t) = \sum_{k=0}^{N-1} I_k \cdot e^{j2\pi k \cdot t/T} si -T_g \le t \le T \qquad \textbf{(2.2)}$$

On:

T_g: Temps de durada del GI.

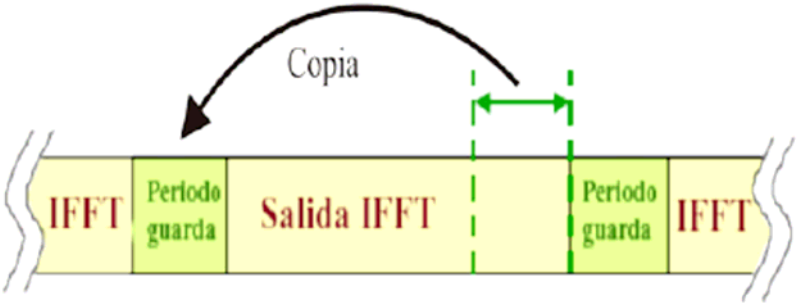


Fig. 2.4 Prefix cíclic

Com ja s'ha dit, les subportadores es poden solapar gràcies a l'ortogonalitat existent entre elles, en canvi, existeixen circumstàncies en que la freqüència de les subportadores pot desplaçar-se lleugerament. Això produeix la pèrdua d'ortogonalitat provocant l'augment de la taxa d'errors en bit (Bit Error RATE –BER-). Aquest fenomen es coneix amb el nom d'interferència entre portadores (Inter Carrier Interference –ICI-).

Els desplaçaments de freqüència poden ocórrer per diferències entre els rellotges del transmissor i receptor, o per efecte Doppler (una variació aparent de la freqüència del senyal degut a l'existència d'un desplaçament relatiu entre el transmissor i el receptor, tot i això, el canvi de freqüència acostuma a ser molt baix per la baixa velocitat a la que es pot desplaçar transmissor o receptor).

El senyal representat per la fórmula **(2.2)** es troba en bandabase i per poder transmetre'l mitjançant l'aire ha de ser portat a una freqüència superior per una portadora d'alta freqüència. El resultat d'aquest procediment és un senyal passabanda representat per la següent equació **(2.3)**:

$$S(t) = \mathrm{Re}\{v(t) \cdot e^{j2\pi fct}\}$$

$$S(t) = \sum_{k=0}^{N-1} \left| I_k \right| \cdot \cos\left[2\pi(fc + k/T) \cdot t + \arg(I_k)\right] \qquad \textbf{(2.3)}$$

On:

fc: Freqüència de la portadora.

2.2.2 Ortogonalitat entre subportadores

Un senyal OFDM compleix una relació matemàtica precisa entre les freqüències de les subportadores. Des d'una visió temporal de la propietat, a l'interval T totes les freqüències tenen exactament un número sencer de cicles. També, el nombre de cicles entre subportadores adjacents difereix exactament en una unitat. La següent figura **(Fig. 2.6)** mostra la representació de les primeres quatre subportadores en el domini temporal.

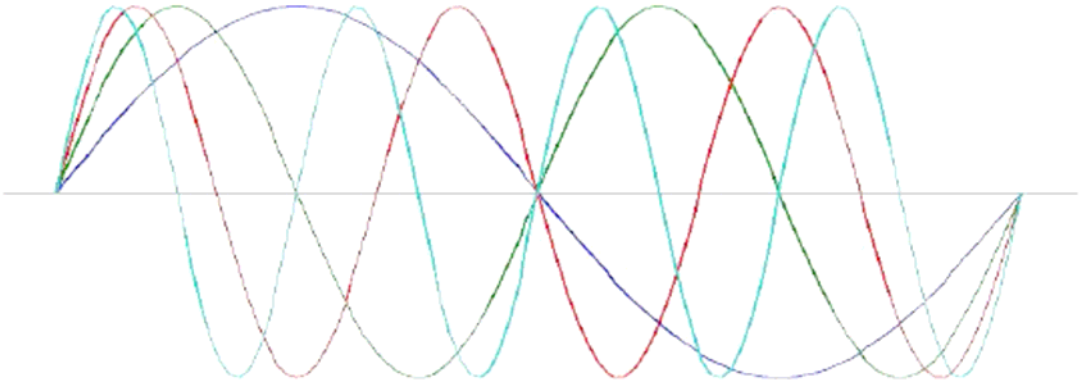


Fig.2.6 Ortogonalitat de les 4 primeres subportadores (domini temporal)

Una manera equivalent d'apreciar la propietat d'ortogonalitat és a partir de la representació freqüencial del senyal. L'espectre del conjunt és el resultat de convolucionar cadascuna de les deltes localitzades a les freqüències de les subportadores amb l'espectre d'un pols rectangular de durada T segons. L'amplitud de l'espectre del pols segueix una funció sinc(5*f*T), amb zeros en totes les freqüències múltiples senceres de 1/T. Llavors, quan una subportadora pren el valor màxim en el domini freqüencial, en el mateix punt, la resta de subportadores s'anul·len, evitant així qualsevol interferència espectral entre elles **(Fig. 2.7)**.

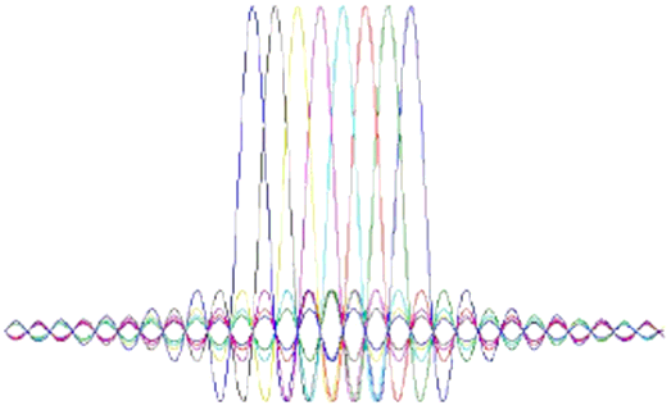


Fig. 2.7 Espectres individuals de 8 subportadores OFDM

2.2.3 Enfinestrar (Windowing)

En un senyal OFDM la transició en la frontera entre el final d'un símbol i el començament del següent pot ser molt abrupta i això provoca components espectrals d'alta freqüència augmentant així l'ample de banda del senyal. Per evitar aquest problema s'utilitzen finestres al començament i al final de cada símbol, les quals permeten augmentar o disminuir la potència del senyal de forma gradual.

Existeixen vàries funcions amb les que es pot portar a terme aquest procediment, com ara el cosinus realçat, Hann, Hamming, Kaiser,…

La figura **(Fig. 2.5)** mostra com la transició entre símbols es fa menys abrupta quan s'utilitza la funció cosinus.

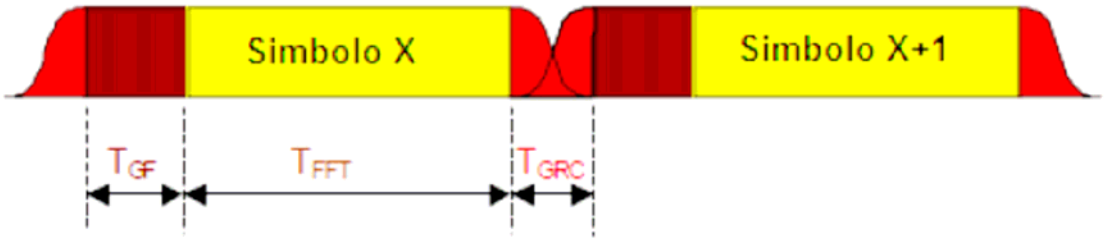


Fig.2.5 Windowing amb la funció cos

2.2.4 Generació d'un símbol OFDM utilitzant IFFT

Weinstein i Ebert van aplicar per primera vegada l'any 1971 la transformada discreta de Fourier (DFT) en el procés de modulació i desmodulació d'una OFDM [8].

En substituir de l'expressió **(2.1)** la variable analògica *t* pels instants de mostreig nT/N_s per n = 0,1,2,...N_s-1, s'obté l'equivalent passabaix discret s(n), **(2.4)**:

$$s(n) = \sum_{k=0}^{N-1} I_k \cdot e^{j2\pi k \cdot n / N_s}, n = 0,1,2,....N_s - 1 \qquad \textbf{(2.4)}$$

Segons aquest resultat, es pot determinar que, llevat un terme constant, la transformació que s'està aplicant a les dades I_k és la Transformada Discreta Inversa de Fourier (IDFT). Aquesta observació va proporcionar un mètode eficient per calcular digitalment l'equivalent passabaix del senyal. Per recuperar les dades I_k del conjunt de mostres s(n), en el procés de desmodulació, només cal realitzar l'operació inversa, en aquest cas la Transformada Discreta de Fourier (DFT). El receptor pot recuperar la informació sense necessitat de filtrat i s'elimina la l'haver de disposar d'un banc d'oscil·ladors per a la generació de les subportadores.

Les transformades DFT i IDFT es realitzen de manera eficient a partir de la utilització dels corresponents algorismes ràpids FFT/IFFT, els quals redueixen el número de multiplicacions complexes necessàries que serien de l'ordre de N^2 a un valor orientatiu de (N/2) $\log_2$ (N) utilitzant l'algorisme de radix-2, gràcies a l'aprofitament de la redundància trobada en les operacions.

2.2.5 Avantatges de la OFDM

L'esquema de modulació OFDM presenta molts avantatges respecte les tècniques que utilitzen una única portadora. A continuació s'enumeren un conjunt.

- OFDM és una estratègia eficient per a combatre els efectes de la dispersió temporal del canal originada per la propagació multicamí.
- Es pot eliminar completament la ISI i la ICI (Inter Carrier Interference) amb les extensions cícliques adequadament dimensionades.
- És una tècnica robusta a les interferències de banda estreta perquè aquestes només afecten a una petita porció de les subportadores.

- Amb un correcte dimensionat de les extensions cícliques, OFDM combina de forma coherent els diferents feixos associats a la propagació multicamí, fet que es tradueix en un guany de diversitat en el receptor.
- La sincronització temporal és flexible i es pot produir dins un marge de valors possibles dins de les extensions cícliques sense degradar les prestacions del sistema.

2.2.6 Inconvenients de la OFDM

Els punts febles d'aquesta tècnica respecte els sistemes basats en una única portadora són bàsicament tres:

- Efectes dels errors de sincronisme freqüencial. La gran sensibilitat als errors de freqüència i al soroll de fase sembla ser la limitació més important. Aquests fenòmens trenquen l'ortogonalitat entre les subportadores i originen interferències –ICI-. Cal habilitar una tècnica d'estimació i correcció de les desviacions de freqüència prou acurada per mantenir el sistema en marges de bon funcionament.
- Efecte dels pics de potència. El senyal OFDM no és constant i en determinades ocasions poden aparèixer pics intensos que porten els amplificadors de radiofreqüència a treballar en zones poc lineals.
- Radiació fora de bandes. Les característiques espectrals del senyal poden no decréixer suficientment ràpid fora de la banda de radiació i originar interferències a altres sistemes, fet que es pot combatre enfinestrant el senyal.

CAPÍTOL 3. IMPLEMENTACIÓ DE LA CAPA FÍSICA (PHY)

3.1 Introducció

Com ja s'ha explicat anteriorment la capa física està dividida en dues subcapes: la PLCP i la PMD.

L'objectiu d'aquest treball final de carrera és implementar totes les funcions que es fan a nivell de bit a la capa física, per tant a la capa PLCP.

Per fer aquest capítol, i en especial, aquest apartat més entenedor es parteix de les figures **(Fig. 3.1** i **Fig. 3.2)**, cadascuna d'elles és un esquema on apareixen les funcions que s'implementen a la capa PLCP, una en emissió i l'altra en recepció.

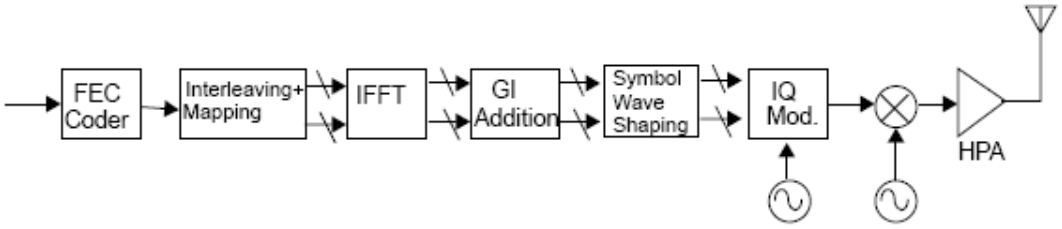


Fig. 3.1 Esquema de l'emissor

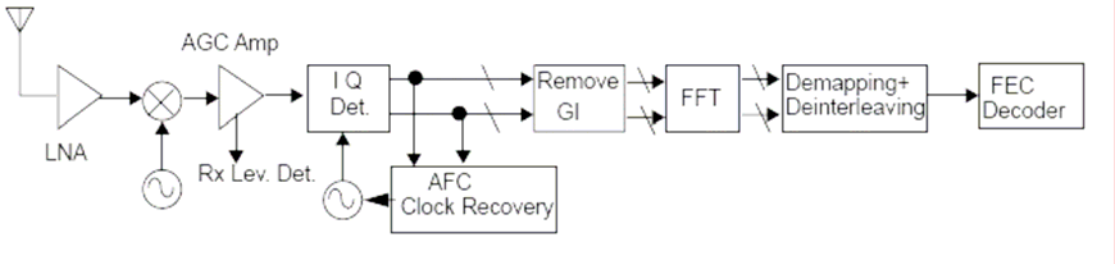


Fig. 3.2 Esquema del receptor

Les funcions a nivell de bit que afecten el transmissor són:

- FEC Coder (Scrambling, Codificació convolucional)
- Interlaving

- Mapping (Mapeig dels bits segons el mapa de modulació a utilitzar)

Les que afecten el receptor són:

- Demapping (Desmapeig dels bits segons el mapa de desmodulació a utilitzar)
- Deinterleaving
- FEC Decoder (Descrambling, Viterbi)

Al capítol anterior s'explicava que una forma senzilla de generar/recuperar símbols OFDM és utilitzant una IFFT/FFT. Degut a la càrrega computacional que requereix una DSP per poder fer aquesta operació, no serà implementada en aquest treball, tot i que com més endavant s'especificarà es tindrà en compte per formar els símbols ODFM (*Veure apartat 3.2.4*).

3.1.1 Format del frame PLCP (PPDU)

La figura **(Fig. 3.3)** mostra el format d'un frame PPDU, és a dir, la concatenació del PLCP Preamble, el PLCP Header, el PSDU (informació realment útil per les capes superiors), els bits de Tail i els bits de Pad.

La capçalera del PLCP (PLCP Header) conté els següents camps: LENGHT, RATE, un bit Reserved, un even Parity bit i el camp SERVICE. En termes de modulació, els camps LENGTH, RATE, Reserved bit, Parity bits i els bits de Tail constitueixen un símbol OFDM anomenat SIGNAL, el qual es transmet amb la modulació més robusta, BPSK i una codificació R=1/2.

El camp SERVICE de la capçalera i el PSDU amb els bits de Tail i Padding, conformen un o més símbols OFDM, anomenat DATA que són transmesos a la velocitat marcada en el camp RATE de la capçalera (PLCP Header). Per tant, per poder processar els símbols DATA és necessari tenir present la capçalera ja que conté informació imprescindible (RATE i LENGTH).

Tots els termes aquí exposats aniran essent detallats durant la resta del document.

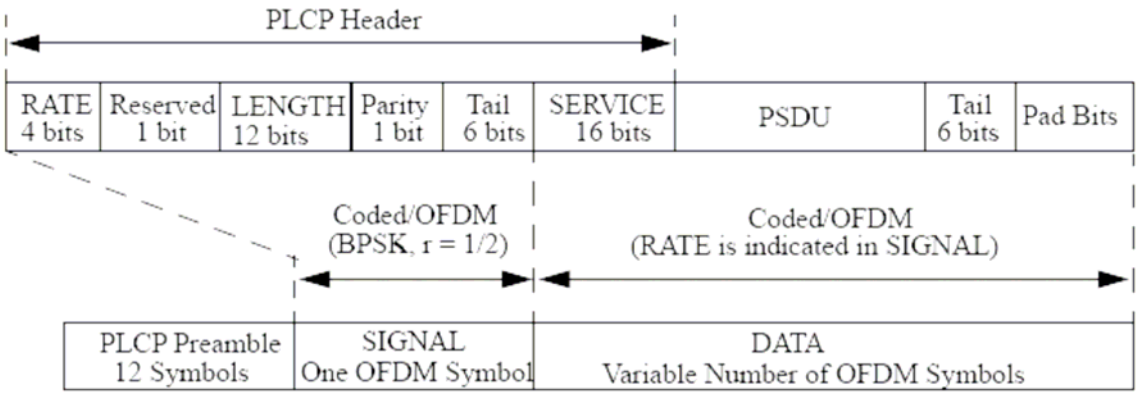


Fig. 3.3 Format d'un frame PLCP

3.1.2 Introducció a la codificació del PPDU

El procés de codificació està composat per diferents passos, tots detallats en els següents apartats. A continuació es mostra un esquema:

a) Construir el camp PLCP Preamble, composat de 10 repeticions d'una "short training sequence", utilitzada per CAG (Control Automàtic de Guany), selecció de diversitat, temps d'adquisició, ajust en freqüència (en el receptor) i dues repeticions d'una "long training sequence" utilitzada per estimar la resposta del canal i fer un ajustament fi de freqüència en el receptor. A això li segueix un interval de guarda.

b) Construir el camp PLCP Header amb els valors RATE, LENGTH i SERVICE que capes superiors hagin proporcionat per aquest frame, omplint els bits corresponents. Els camps RATE i LENGTH del PLCP Header són codificats amb un codi convolucional R=1/2 i després mapats amb BPSK en un únic símbol OFDM, anomenat SIGNAL al qual s'afegeixen 6 bits de Tail per tal de tornar el codificador convolucional a l'estat inicial. La codificació del camp SIGNAL en un símbol OFDM segueix els mateixos passos de codificació convolucional, Interleaving, modulació BPSK i IFFT que si del camp DATA es tractés, només que a una velocitat de 6 Mbps i tenint en compte que SIGNAL no passa per l'Scrambler.

c) Calcular gràcies al camp RATE el número de bits del camp DATA per símbol OFDM (N_{DBPS}), la taxa de codificació (coding RATE –R-), el número de bits en cada subportadora OFDM (N_{BPSC}), i el número de bits codificats per símbol OFDM (N_{CBPS}).

d) Afegir el PSDU al camp SERVICE. Afegir els sis bits de Tail més els de Padding per tal que la longitud final sigui múltiple de N_{DBPS}. La cadena de bits resultants constitueix la part DATA del paquet.

e) Inicialitzar l'scrambler amb una llavor pseudo-aleatòria diferent de 0 i fer-li XOR amb la cadena de bits de DATA.

f) Reemplaçar els sis bits de Tail, més els de Padding scrambled per sis zeros. (Aquests bits retornaran a l'estat 0 el codificador convolucional).

g) Codificar els bits del camp DATA scrambled amb un codificador convolucional de taxa R=1/2. Per algunes RATE caldrà eliminar alguns bits segons els patrons marcats per la tècnica de puncturing.

h) Dividir la cadena de bits codificats en grups de N_{CBPS}. A cada grup, realitzar interleaving (reordenació de bits) d'acord amb les regles corresponents a la RATE utilitzada.

I) Dividir el resultat de bits codificats i interleaved en grups de N_{CBPS} bits. Per cada grup de bits, convertir el grup en un número complex d'acord a la taula de modulació escollida (BPSK, QPSK, 16QAM,…)

j) Dividir la cadena de números complexos en grups de 48. Cada grup serà associat a un símbol OFDM. En cada grup, els números complexos seran numerats del 0 al 47 i mapats en les portadores del símbol OFDM numerades de -26 a -22, -20 a -8, -6 a -1, 1 a 6, 8 a 20 i 22 a 26. Les subportadores -21, -7, 7 i 21 són saltades i per tant, usades per insertar subportadores pilot. La subportadora 0, associada al centre de freqüència és omesa i mapada a 0.

k) Quatre subportadores són inserides com pilots en les posicions -21, -7, 7 i 21. El número total de subportadores és 52 (48+4).

NOTA: Els passos següents no estan implementats a l'aplicació programada en C per no formar part dels objectius del projecte.

l) Per cada grup de subportadores -26 a 26, convertir les subportadores a domini temporal utlitzant una IFFT. Afegir a la transformada de Fourier el prefix cíclic, formant un GI i aplicar Windowing.

m) Concatenar els símbols OFDM, començant pel símbol SIGNAL.

n) Traslladar la forma d'ona obtinguda a banda base a la freqüència central d'emissió.

3.1.3 Paràmetres relacionats amb el camp RATE

A continuació es presenta una taula **(Taula 3.1)**, que relaciona diferents paràmetres associats al camp RATE que afectaran més endavant a altres apartats com ara el codificador convolucional, l'interleaving o el mapatge de símbols entre d'altres.

Taula 3.1. Paràmetres associats al camp RATE

Data rate (Mbits/s)	Modulation	Coding rate (R)	Coded bits per subcarrier (N_{BPSC})	Coded bits per OFDM symbol (N_{CBPS})	Data bits per OFDM symbol (N_{DBPS})
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24	16-QAM	1/2	4	192	96
36	16-QAM	3/4	4	192	144
48	64-QAM	2/3	6	288	192
54	64-QAM	3/4	6	288	216

3.1.4 Paràmetres temporals associats al frame OFDM PLCP

A continuació es presenta una taula **(Taula 3.2)** amb els valors temporals associats al frame OFDM PLCP.

Taula 3.2 Paràmetres temporals associats al frame OFDM PLCP

Parameter	Value
N_{SD}: Number of data subcarriers	48
N_{SP}: Number of pilot subcarriers	4
N_{ST}: Number of subcarriers, total	52 ($N_{SD} + N_{SP}$)
Δ_F: Subcarrier frequency spacing	0.3125 MHz (=20 MHz/64)
T_{FFT}: IFFT/FFT period	3.2 μs ($1/\Delta_F$)
$T_{PREAMBLE}$: PLCP preamble duration	16 μs ($T_{SHORT} + T_{LONG}$)
T_{SIGNAL}: Duration of the SIGNAL BPSK-OFDM symbol	4.0 μs ($T_{GI} + T_{FFT}$)
T_{GI}: GI duration	0.8 μs ($T_{FFT}/4$)
T_{GI2}: Training symbol GI duration	1.6 μs ($T_{FFT}/2$)
T_{SYM}: Symbol interval	4 μs ($T_{GI} + T_{FFT}$)
T_{SHORT}: Short training sequence duration	8 μs ($10 \times T_{FFT}/4$)
T_{LONG}: Long training sequence duration	8 μs ($T_{GI2} + 2 \times T_{FFT}$)

3.1.5 Especificacions de la IFFT

La forma més ràpida i habitual d'implementar la transformada inversa de Fourier, com ja s'ha comentat en apartats anteriors, és mitjançant l'algorisme Fast Fourier Transform (IFFT).

Per aquest projecte s'utilitzarà, una IFFT de 64 punts. Els coeficients 1 al 26 estan mapats en el mateix número d'entrada del mòdul IFFT, mentre que els coeficients -26 a -1 són copiats a les entrades del mòdul IFFT 38 a 63. La resta d'entrades, 27 a 37 i la 0 (dc) són mapades a 0. La figura **(Fig. 3.4)** mostra un exemple.

Després de fer la IFFT, la sortida és estesa cíclicament a la longitud desitjada.

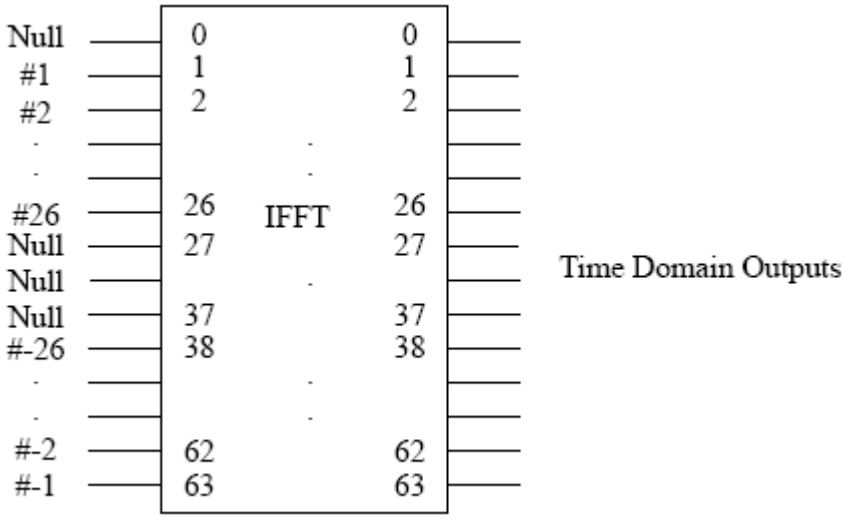


Fig. 3.4 Entrades i sortides del mòdul IFFT

3.2 PLCP Preamble (SYNC)

El camp PLCP Preamble és utilitzat per sincronització. Consisteix en 10 símbols curts (short symbols) i 2 símbols llargs (long symbols) que són mostrats a la figura **(Fig. 3.5)**.

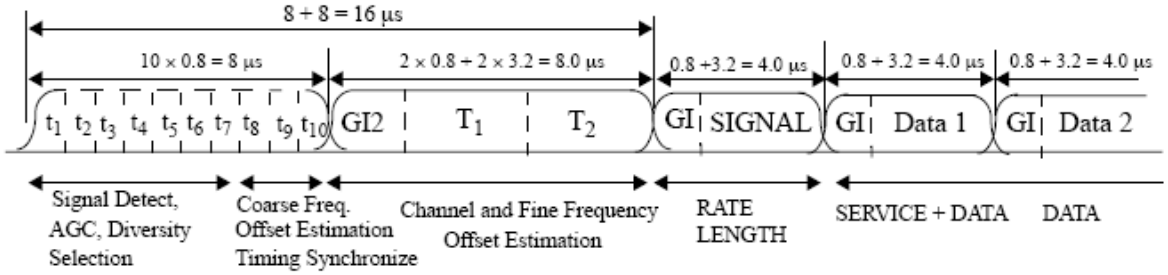


Fig. 3.5 Estructura OFDM training

La figura **(Fig. 3.5)** mostra el PLCP Preamble on t1 a t10 són els short symbols i T1 i T2 són els long symbols. El camp PLCP Preamble és seguit del camp SIGNAL i del camp DATA.

Degut a que cada paquet OFDM porta el PLCP Preamble i que aquests valors són constants podem evitar la càrrega computacional que provoca l'haver de calcular

per cada paquet els valors d'aquest camp. (Veure les especificacions 802.11a per aconseguir aquests valors ja calculats).

El codi desenvolupat en C no incorpora la creació del PLCP Preamble ja que és un valor constant que pot afegir-se directament en el mòdul IFFT.

3.3 El camp SIGNAL

Al PLCP Preamble el segueix el camp SIGNAL, el qual conté els camps RATE i LENGTH principalment. El camp RATE conté informació sobre el tipus de modulació i la taxa del codificador convolucional de la resta del paquet (camp DATA). La codificació del camp SIGNAL es fa en únic símbol OFDM mitjançant una modulació BPSK de les subportadores utilitzant una taxa R=1/2 al codificador convolucional.

El procés de codificació, que inclou codificació convolucional, interleaving, mapeig de la modulació, inserció de canals pilot i modulació OFDM segueix els mateixos passos que per un símbol DATA a una velocitat de 6 Mbps a excepció que el camp SIGNAL no passa per l'scrambler.

El camp SIGNAL està composat per 24 bits com es mostra a la figura **(Fig. 3.6)**. Els 4 bits, 0 a 3, corresponen al camp RATE. El bit 4 és un bit reservat per un ús futur. Els bits 5-16 codifiquen la longitud del PSDU (el bit de menys pes LSB és el primer en ser transmès). El bit 17 indica la paritat (even) i del 18 al 23 es troben els bits de Tail a '0' que retornaran el codificador convolucional a l'estat inicial.

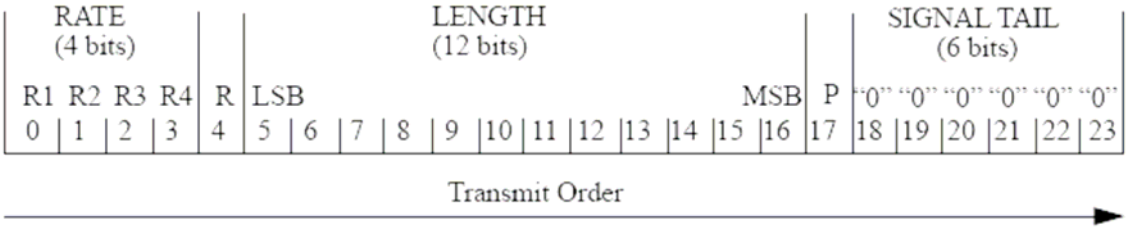


Fig. 3.6 Bits del camp SIGNAL

3.3.1 El camp RATE

La següent taula **(Taula 3.3)** mostra la correspondència entre la codificació dels bits R1-R4 amb la velocitat associada.

Taula 3.3 Codificació de la velocitat RATE

Rate (Mbits/s)	R1–R4
6	1101
9	1111
12	0101
18	0111
24	1001
36	1011
48	0001
54	0011

3.3.2 El camp LENGTH

El camp LENGTH està composat per 12 bits que indiquen el número de bytes que té el PSDU. El LSB és primer en ser transmès.

3.3.3 El camp Reserved, Parity i Tail BITS

El bit 4 (Reserved) no s'utilitza actualment i es mapa a 0. El bit 17 correspon a l'even Parity dels bits 0-16. Els bits 18-23 constitueixen els bits de Tail del camp SIGNAL i són 6 '0'.

3.4 El camp DATA

El camp DATA conté el camp SERVICE, el PSDU, els bits de Tail i els bits de Pad en cas de ser necessaris.

3.4.1 El camp SERVICE

Aquest camp té 16 bits, 0-15. El bit 0 és el primer en ser transmès. Els bits 0-6 estan a 0 i s'utilitzen per sincronitzar el descrambler en el receptor. La resta de bits no tenen actualment utilitat i estan a 0 **(Fig. 3.7)**.

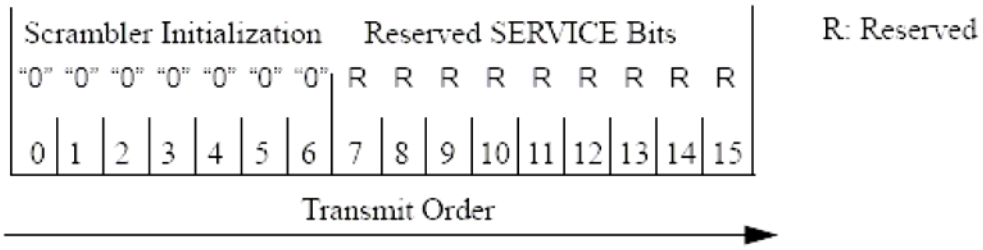


Fig.3.7 Bits del camp SERVICE

3.4.2 Els bits de Tail

El camp de Tail s'omple amb sis '0', els quals són requerits per tornar el codificador convolucional a l'estat inicial. Aquest procés millora la probabilitat d'error del descodificador convolucional. Com que aquests sis '0' passaran per l'scrambler i el més probable és que aquests valors quedin modificats, cal assegurar-se que del transmissor sempre sortiran sis '0' modificant si cal, el valor resultant de l'scrambler.

3.4.3 Els bits de Pad

El número de bits en el camp DATA ha de ser múltiple de N_{CBPS}, número de bits codificats en un símbol OFDM (48, 96, 192 o 288 bits). Per aconseguir-ho, al missatge s'afegeixen els 0's necessaris fins aconseguir que el missatge sigui múltiple de N_{DBPS}, número de bits de dades per símbol OFDM. Com a mínim 6 bits seran afegits (els de Tail).

El número de símbols OFDM, N_{SYMB}, el número de bits en el camp DATA, N_{DATA}, i el número de bits de Pad s'obtenen amb les fórmules **(3.1, 3.2, 3.3)** respectivament.

$$N_{SYM} = \text{Ceiling}\left((16 + 8 \times \text{LENGTH} + 6)/N_{DBPS}\right) \quad \textbf{(3.1)}$$

$$N_{DATA} = N_{SYM} \times N_{DBPS} \quad \textbf{(3.2)}$$

$$N_{PAD} = N_{DATA} \ (16 + 8 \times LENGTH + 6) \quad \textbf{(3.3)}$$

La funció celing (.) retorna l'enter més petit, menor o igual, que l'argument que rep.

Tot i que els bits de Tail i els de Pad hauran de ser al final 0's, cal que aquests segueixin de principi a fi, tots els processos requerits pel camp DATA.

3.5 Funcions per codificar/descodificar el frame PLCP

Els següents apartats tenen com a objectiu explicar en detall totes aquelles funcions que l'aplicació implementa per tal d'aconseguir els símbols finals a transmetre a capes inferiors (en mode TX) o superiors (en mode RX).

3.5.1 PLCP DATA scrambler i descrambler

Una de les funcions que s'han de desenvolupar en el bloc de FEC coder i FEC decoder és la de codificar i descodificar respectivament la seqüència de bits a enviar/rebre. La funció de l'scrambler és la d'evitar llargues seqüències de '1' o '0' que fan la comunicació més vulnerable a errors.

NOTA: Per no confondre codificació convolucional i la codificació realitzada en aquest apartat, per a aquesta última utilitzarem el mot anglès, scrambler.

El camp DATA, format pels camps SERVICE, PSDU, Tail i Pad, ha de ser scrambled amb un scrambler síncron de longitud 127. Els octets del PSDU són disposats en sèrie, bit 0 primer i bit 7 l'últim. L'scrambler utilitza el polinomi generador S(x) que segueix la fórmula **(3.4)** i la figura **(Fig. 3.8)**.

$$S(x) = x^7 + x^4 + 1 \quad \textbf{(3.4)}$$

La seqüència generada de 127 bits utilitzada per l'scrambler ha de ser en un principi 00001110 11110010 11001001 00000010 00100110 00101110 10110110 00001100 11010100 11100111 10110100 00101010 11111010 01010001 10111000 1111111.

Aquesta seqüència serà utilitzada tan per codificar com descodificar.

Els 7 LSBs del camp SERVICE han d'estar a '0' abans de fer l'scrambling per poder fer l'estimació de l'estat inicial de l'scrambler en el receptor.

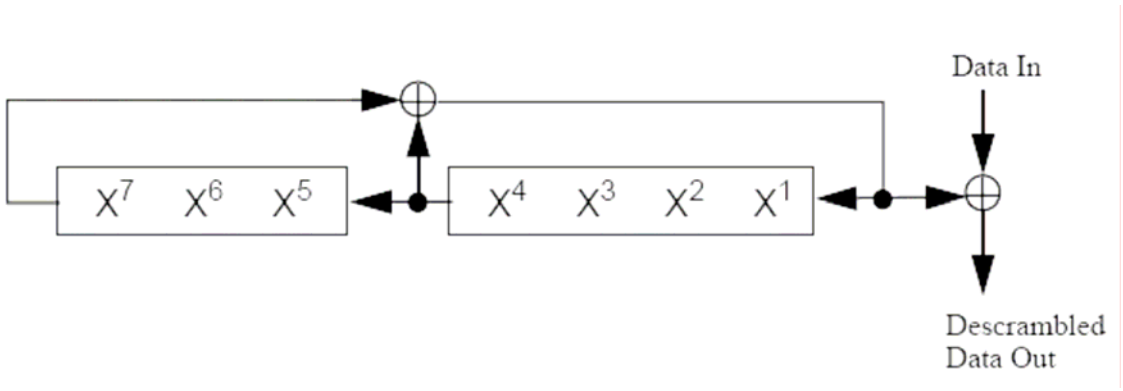


Fig.3.8 Scrambler i Descrambler

3.5.2 Codificador convolucional

Un codificador convolucional és un codi utilitzat per corregir errors (en anglès forward error correction) en el qual un grup de M bits d'informació es codifiquen mitjançant algorismes de codificació i es transformen en un conjunt de N bits. S'ha de complir que M<N degut a que durant el procés de codificació s'afegeix informació redundant que permetrà al receptor corregir els errors que apareguin.

El coeficient M/N es coneix amb el nom de taxa de codi (en anglès code RATE) i pren valors entre 0 i 1. Quan la taxa de codi pren valors baixos el codi es fa més robust, és a dir, existeix major redundància en la informació transmesa i per tant augmenta la probabilitat de poder corregir els bits erronis, en canvi això redueix la velocitat efectiva de transmissió degut a l'augment en el tamany del número de bits a enviar. Si pel contrari la taxa és alta, es fa més difícil corregir els bits erronis però s'augmenta la velocitat efectiva de transmissió.

El codi convolucional no és part de la tècnica OFDM en sí, però és molt comú la seva implementació en sistemes OFDM, d'aquesta manera s'evita que interferències en una banda estreta del canal o que el soroll d'aquest provoquin increments considerables del BER.

Els sistemes OFDM que inclouen el codi convolucional com a mecanisme de correcció d'errors es coneix com COFDM (Coded OFDM).

El camp DATA, composat pels camps SERVICE, PSDU, Tail i Pad, ha de ser codificat convolucionalment amb un codi de taxa R=1/2, 2/3 o 3/4, segons la velocitat RATE seleccionada (Veure **Taula 3.3**).

El codificador convolucional utilitza els polinomis generadors $g_0 = 133_8$ and $g_1 = 171_8$ amb una taxa R=1/2. (Veure **Fig. 3.9**).

El bit anomenat “A” ha de sortir del codificador abans que el “B”.

Velocitats superiors s’aconsegueixen utilitzant “puncturing”. Puncturing és el procés pel qual s’ometen alguns dels bits codificats pel codificador convolucional, reduint així el número de bits a transmetre i augmentant la velocitat fins aconseguir la marcada al camp RATE. El receptor inserirà ‘0’ en el lloc dels bits eliminats, intentant recuperar els bits originals. Les figures **(Fig. 3.10 i Fig. 3.11)** mostren el patró a seguir per realitzar aquesta tècnica.

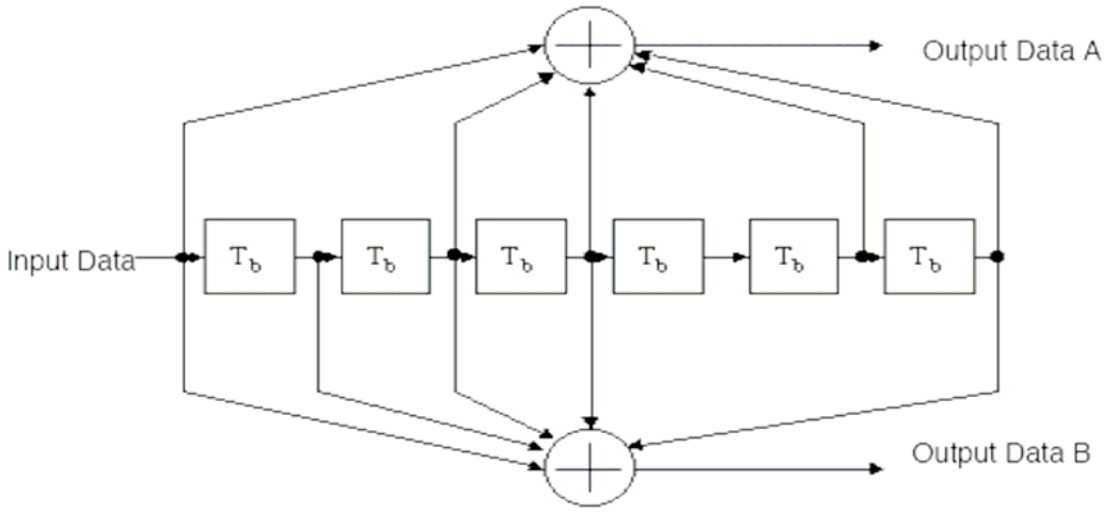


Fig. 3.9 Codificador convolucional

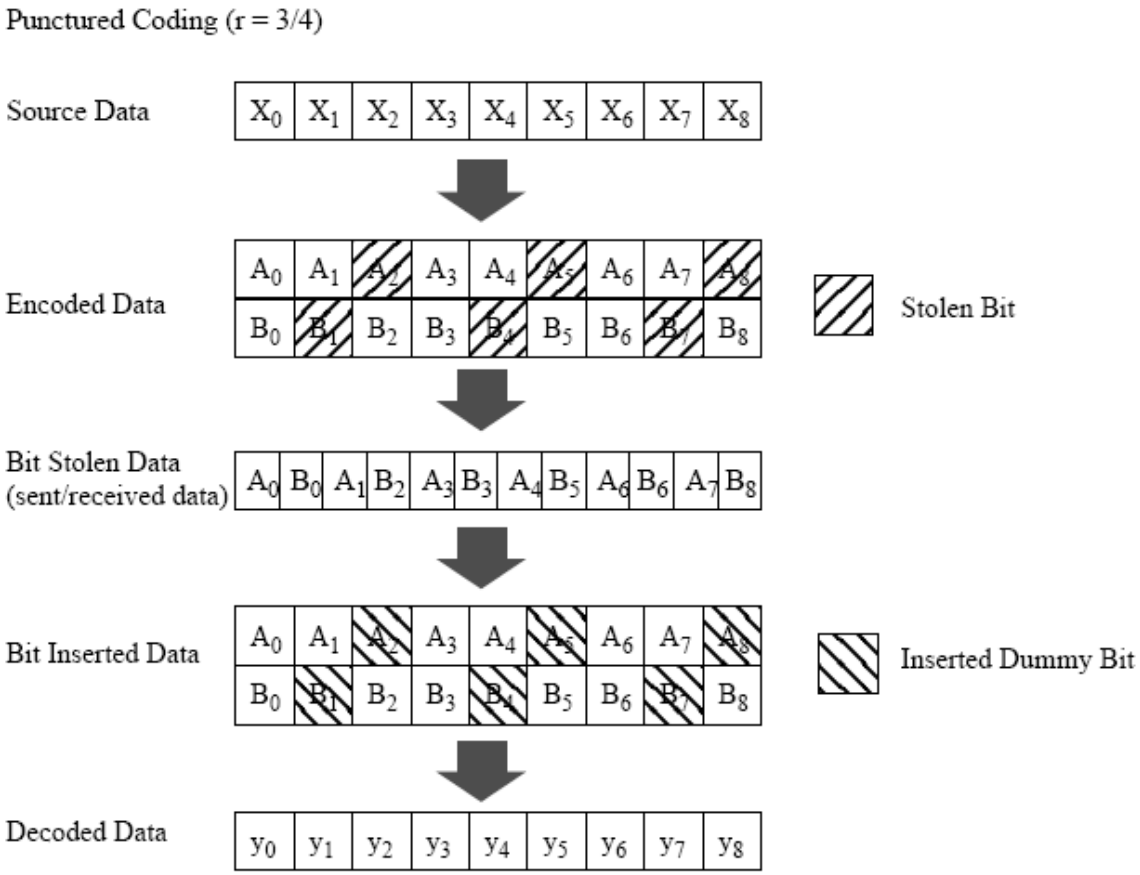


Fig. 3.10 Puncturing per R=3/4

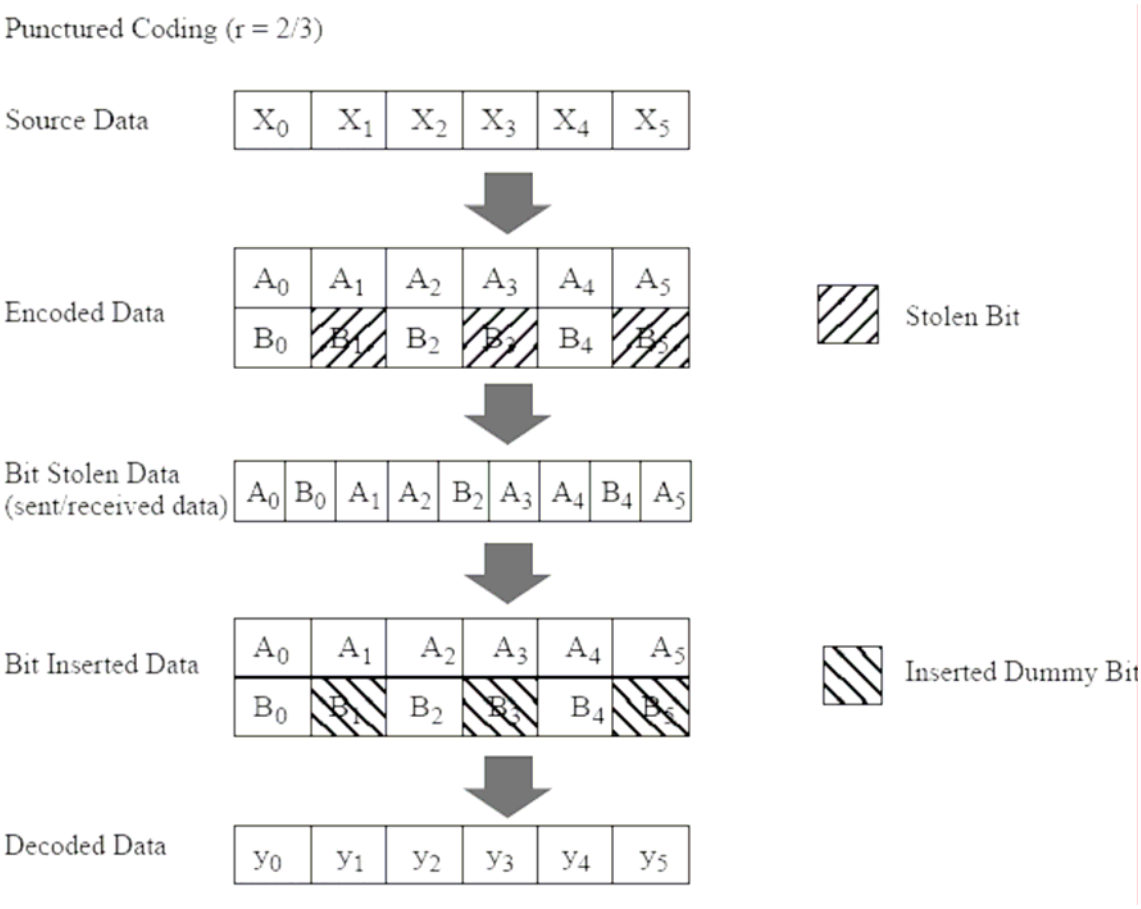


Fig. 3.11 Puncturing per R=2/3

3.5.3 Descodificador convolucional de VITERBI

Per desfer la codificació convolucional realitzada en transmissió es recomana utilitzar un descodificador de Viterbi.

3.5.3.1 Funció de versemblança

Donada una única observació **y** en el receptor, existeix una seqüència **x** enviada, que anomenem $x_{màx}$, dins del conjunt de seqüències enviades possibles de **x** que fa màxima la funció de versemblança. És a dir, la seqüència $x_{màx}$ és la que té més probabilitat d'haver generat l'observació **y**.

Per un codificador convolucional determinat, s'observa que a cada node hi conflueix més d'un branca. L'algorisme de Viterbi permetrà escollir en cada temps

tj quina és la branca que acumula el camí (entès com a un conjunt de branques) més versemblant.

3.5.3.2 Definició de l'algorisme de Viterbi

Desenvolupat per Andrew J. Viterbi l'any 1967 [9]. Aplica la funció de màxima versemblança tot reduint la càrrega computacional, ja que utilitza l'estructura del diagrama de Trellis.

L'algorisme de Viterbi permet descartar els camins que no tenen possibilitats de ser considerats de màxima versemblança: Acció de podar. Així, el còmput queda reduït al conjunt de camins supervivents.

3.5.4 Interleaving

En un canal de comunicacions sense fils acostumen a ocórrer esvaïments en determinades freqüències, aquest fenomen aleatori provoca canvis en la amplitud de les subportadores d'un senyal OFDM, i per tant el receptor rep diferents nivells de potència en cada subportadora. En alguns casos la potència d'un grup de subportadores és tan baixa que no permet identificar la informació (bits) que conté. Per evitar que aquest fenomen produeixi ràfegues d'errors que no puguin ser corregits, s'intercanvia la posició dels bits abans de ser transmesos, és a dir s'ordenen els bits de tal manera que no existeixin bits continus en subportadores adjacents.

El procediment abans descrit conegut com Interleaving permet que els bits erronis ocorrin aleatòriament i separats uns dels altres. Els bits erronis que apareixen de forma aïllada es poden recuperar gràcies al codi convolucional, mentre que les ràfegues d'errors requereixen retransmissió.

Tots els bits codificats (pel codificador convolucional) són interleaved, reordenats, per un bloc interleaver amb un tamany corresponent al número de bits d'un símbol OFDM, N_{CBPS}. L'interleaver està definit per dues permutacions. La primera assegura que els bits adjacents codificats són mapats en subportadores no adjacents. La segona permutació assegura que els bits adjacents codificats són mapats alternativament en bits més o menys significatius de la constel·lació i per tant, que no s'enviïn seguits massa bits LSB.

Anomenarem **k** l'índex del bit codificat abans de la primera permutació; **i** a l'índex després de la primera permutació i abans de la segona, i **j** l'índex després de la segona permutació i just abans del mapatge de la modulació.

La primera permutació està definida com **(3.5)**:

$$i = (N_{CBPS}/16)(k \bmod 16) + floor(k/16), k = 0,1,..., N_{CBPS} - 1 \quad \textbf{(3.5)}$$

La funció floor(.) retorna l'enter més gran que no excedeix l'argument donat.

La segona permutació segueix la fórmula **(3.6)**:

$$j = s \cdot floor(i/s) + (i + N_{CBPS} - floor(16 \cdot i / N_{CBPS})) \bmod s, i = 0,1,...N_{CBPS} - 1 \quad \textbf{(3.6)}$$

El valor de **s** és determinat pel número de bits codificats per subportadora, N_{BPSC}, i segueix la fórmula **(3.7)**:

$$s = \max(N_{BPSC}/2,1) \quad \textbf{(3.7)}$$

3.5.5 Deinterleaving

El deinterleaving segueix la funció inversa a l'interleaving i està definit per dues permutacions.

Anomenem **j** l'índex del bit original abans de la primera permutació, **i** a l'índex després de la primera permutació i abans de la segona, i **k** l'índex després de la segona permutació, just abans d'entregar els bits al descodificador convolucional Viterbi.

La primera permutació segueix l'equació **(3.9)**:

$$i = s \cdot floor(j/s) + (j + floor(16 \cdot j / N_{CBPS})) \bmod s, j = 0,1...N_{CBPS} - 1 \quad \textbf{(3.9)}$$

Nota: **s** segueix la mateixa equació que **(3.7)**.

La segona permutació **(3.10)**:

$$k = 16 \cdot i - (N_{CBPS} - 1) floor(16 \cdot i / N_{CBPS}), i = 0,1,...N_{CBPS} - 1 \quad \textbf{(3.10)}$$

3.5.6 Mapeig i desmapeig de les subportadores

Les subportadores OFDM són modulades utilitzant BPSK, QPSK, 16-QAM o 64-QAM, depenent de la RATE demanada (Veure **Taula 3.1**).

La cadena de bits codificats i reordenats (interleaved) són dividits en grups de N_{BPSC} (1, 2, 4 o 6) bits i convertits en números complexos representat punts de la constel·lació de la BPSK, QPSK, 16-QAM o 64-QAM. La conversió ha de ser feta amb codi Gray i seguint els mapatges de les taules **(Taules 3.5** a **3.8)**, on el bit d'entrada b_0 és el primer de la cadena. Els valors de sortida, **d**, s'obtenen multiplicant el resultat (I+jQ) per un valor de normalització K_{MOD} com es descriu a l'equació **(3.11)**.

$$d = (I + jQ) \times K_{MOD} \qquad \textbf{(3.11)}$$

El factor de normalització, K_{MOD}, depèn de la modulació utilitzada (Veure **Taula 3.4**).

Cal fixar-se que el tipus de modulació pot ser diferent del camp SIGNAL al camp DATA. L'objectiu del factor de normalització és aconseguir la mateixa mitjana de potència per tots els mapatges.

Taula 3.4 Factors de normalització K_{MOD}

Modulation	K_{MOD}
BPSK	1
QPSK	$1/\sqrt{2}$
16-QAM	$1/\sqrt{10}$
64-QAM	$1/\sqrt{42}$

Per una BPSK, b_0 determina el valor en fase **I** com il·lustra la taula **(Taula 3.5).**
Per una QPSK, b_0 determina el valor **I** i b_1 el valor **Q**, quadratura **(Taula 3.6).**
Per una 16-QAM b_0b_1 determina el valor en fase i b_2b_3 la quadratura **(Taula 3.7).**
Per una 64-QAM $b_0b_1b_2$ determina la fase i $b_3b_4b_5$ la quadratura **(Taula 3.8).**

Taula 3.5 Codificació BPSK

Input bit (b_0)	I-out	Q-out
0	–1	0
1	1	0

Taula 3.6 Codificació QPSK

Input bit (b_0)	I-out
0	–1
1	1

Input bit (b_1)	Q-out
0	–1
1	1

Taula 3.7 Codificació 16-QAM

Input bits (b_0 b_1)	I-out
00	–3
01	–1
11	1
10	3

Input bits (b_2 b_3)	Q-out
00	–3
01	–1
11	1
10	3

Taula 3.8 Codificació 64-QAM

Input bits (b_0 b_1 b_2)	I-out
000	–7
001	–5
011	–3
010	–1
110	1
111	3
101	5
100	7

Input bits (b_3 b_4 b_5)	Q-out
000	–7
001	–5
011	–3
010	–1
110	1
111	3
101	5
100	7

3.5.7 Canals pilot

En cada símbol OFDM, quatre de les subportadores són dedicades a senyals pilots per tal de fer més robusta la detecció correcta dels símbols i evitar soroll. Aquests senyals pilot són inserits a les subportadores -21, -7, 7 i 21. Els canals pilot són modulats amb BPSK seguint una seqüència pseudo binària per evitar la generació de línies espectrals.

La polaritat de les subportadores pilot és controlada per la seqüència pn que té una extensió cíclica dels 127 elements següents:

p0..126v = {1,1,1,1, -1,-1,-1,1, -1,-1,-1,-1, 1,1,-1,1, -1,-1,1,1, -1,1,1,-1, 1,1,1,1, 1,1,-1,1,1,1,-1,1, 1,-1,-1,1, 1,1,-1,1, -1,-1,-1,1, -1,1,-1,-1, 1,-1,-1,1, 1,1,1,1, -1,-1,1,1,-1,-1,1,-1, 1,-1,1,1, -1,-1,-1,1, 1,-1,-1,-1, -1,1,-1,-1, 1,-1,1,1, 1,1,-1,1, -1,1,-1,1,-1,-1,-1,-1, -1,1,-1,-1,1, 1,-1,1,-1, 1,1,1,-1, -1,1,-1,-1, -1,1,1,1, -1,-1,-1,-1, -1,-1,-1}

L'element p0 multiplica les subportadores pilot del símbol SIGNAL, mentre que els elements a partir de p1 multipliquen els símbols DATA.

CAPÍTOL 4. Descripció de l'aplicació "MyWlan"

4.1 Introducció

Un dels objectius més interessants d'aquest TFC és la realització d'un programa en C configurable sobre una DSP que implementi les operacions que es realitzen a la capa PLCP.

Abans de fer la implementació final a la DSP s'ha utilitzat l'eina Comunications Manager –CM- per tal d'executar en temps real el procés de TX i RX, dividint l'aplicació "MyWlan" en diferents seccions. L'apartat 4.2 conté una introducció al CM.

En aquest capítol es presenta el funcionament del Comunications Manager i de l'aplicació "MyWlan".

4.2 Comunications Mannager (CM)

4.2.1 Introducció

CM és un programa que possibilita la integració de les diferents peces que composen una aplicació seguint un conjunt de regles. A l'hora, CM ofereix la possibilitat de descentralitzar les peces de l'aplicació i fer-les córrer en diferents equips connectats en xarxa.

CM pot executar-se sota qualsevol sistema operatiu derivat de Unix. Durant el període de prova s'ha executat sota Linux Mandriva.

En les següents seccions es descriuen les característiques bàsiques del CM aplicades a aquest TFC.

4.2.2 CM Funcions

CM proporciona un conjunt de funcions independents al Sistema Operatiu per controlar els diferents mòduls (peces) que conformen l'aplicació [11].

4.2.3 Procés de comunicació interna

Cada mòdul és considerat com un procés, independentment de que el Sistema Operatiu el tracti així o no. Per passar dades d'un procés a un altre CM ofereix un conjunt de funcions per tal de fer aquest intercanvi d'informació entre màquines. [11].

4.2.4 Paràmetres de configuració

Els mòduls no poden llegir de cap medi físic, per tant, qualsevol paràmetre de configuració ha de ser facilitat mitjançant el CM. Durant el procés d'inicialització cada mòdul llegirà els paràmetres de configuració, no estant permeses configuracions externes, amb l'únic objectiu de centralitzar el procés de configuració i gestió.

4.2.5 Execució de l'aplicació

CM controla que tots els mòduls estiguin correctament registrats i inicialitzats, que funcionin, i que es tanquin correctament. Això és possible gràcies a un procés de monotorització continu que detecta qualsevol error en els mòduls donant informació a l'usuari d'aquest.

Cal recordar que només CM està autoritzat a interactuar amb el Sistema Operatiu durant l'execució del programa, per tant, cap mòdul pot accedir al sistema directament (per exemple per demanar accés a memòria).

4.2.6 Control del temps

Cada mòdul s'executa periòdicament i és CM qui assigna CPU per cada interval de temps definit, no ho pot fer el propi mòdul. CM verifica que l'assignació de memòria es fa correctament en cada període de temps.

4.3 Estructura de l'aplicació "MyWlan" sobre CM

L'aplicació implementada "MyWlan" consta de quatre mòduls que realitzen les funcions de 1) generació de dades procedents de capes superiors (capa MAC), 2) processament de les dades anteriors i enviament per un canal simulat, 3) recepció de dades del canal i processament d'aquestes, 4) entrega de dades finals a capes superiors (capa MAC).

Per tant només el segon i el tercer mòdul són peces de l'estàndard, essent la resta peces de suport que simulen altres capes no implementades al projecte.

Tot i que els quatre mòduls s'executen alhora, el resultat de l'últim mòdul depèn del correcte funcionament de tots els anteriors, en cas contrari, els errors no detectables s'aniran acumulant.

Els següents apartats fan una primera aproximació a l'estructura de l'aplicació "MyWlan", però no obstant s'anirà detallant el funcionament durant la resta de document.

4.3.1 Primer mòdul: capa MAC.

La funció d'aquest mòdul és simular la capa MAC. Per realitzar-ho s'ha implementat un paquet d'exemple que serà el que es processi cíclicament.

Degut a que aquest paquet prové d'una capa superior emulada i per l'explicat a l'apartat 3.1.2, aquest haurà de tenir informació de la velocitat (RATE), longitud del PSDU (LENGHT), el bit Reserved i el propi PSDU. Tota aquesta informació passarà al següent mòdul utilitzant les funcions corresponents del CM que gestionarà l'enviament del paquet.

El format del paquet de prova d'aquest primer mòdul és el següent:

RATE (4b) + Reserved (1b) + LENGHT PSDU (12b) + PSDU (màx. 4095b).

4.3.2 Segon mòdul: 802.11 TX

Aquest mòdul és el programa en sí de processament de dades per l'enviament de paquets al canal.

Rep les dades necessàries mitjançant el CM del primer mòdul (capa MAC) i a partir d'aquestes genera el PPDU final a l'espera de passar a la capa PMD que afegeixi el PLCP Preamble, realitzi la IFFT i enviï els símbols corresponents pel canal (veure 3.1.2).

Aquest mòdul enviarà al següent (802.11 RX) mitjançant el canal emulat, el primer símbol, SIGNAL, tan aviat el tingui processat.

Una vegada el símbol SIGNAL sigui enviat continuarà processant dades i cada vegada que generi un símbol OFDM del camp DATA l'enviarà pel canal.

4.3.3 Tercer mòdul: 802.11 RX

Aquest mòdul s'encarrega de la recepció dels símbols OFDM (una vegada desfeta la IFFT i eliminat el PLCP Preamble) i de la descodificació de la informació.

Ja que d'acord a les especificacions del 802.11 el PPDU s'ha de tractar a nivell sencer, fins que no s'hagin rebut tots els símbols procedents d'un paquet complet (format pel símbol SIGNAL més els símbols DATA) no s'obtindran les dades descodificades per tal d'enviar al següent i últim mòdul (PSDU), la informació més rellevant, el PSDU.

4.3.4 Quart mòdul: PSDU

L'últim mòdul s'encarrega de rebre el PSDU que s'ha descodificat, i que si tot ha anat bé ha de coincidir amb el d'exemple introduït al primer mòdul.

4.3.5 Esquema general de mòduls

La figura **(Fig. 4.1)** mostra un esquema amb els diferents blocs que conformen l'aplicació "MyWlan".

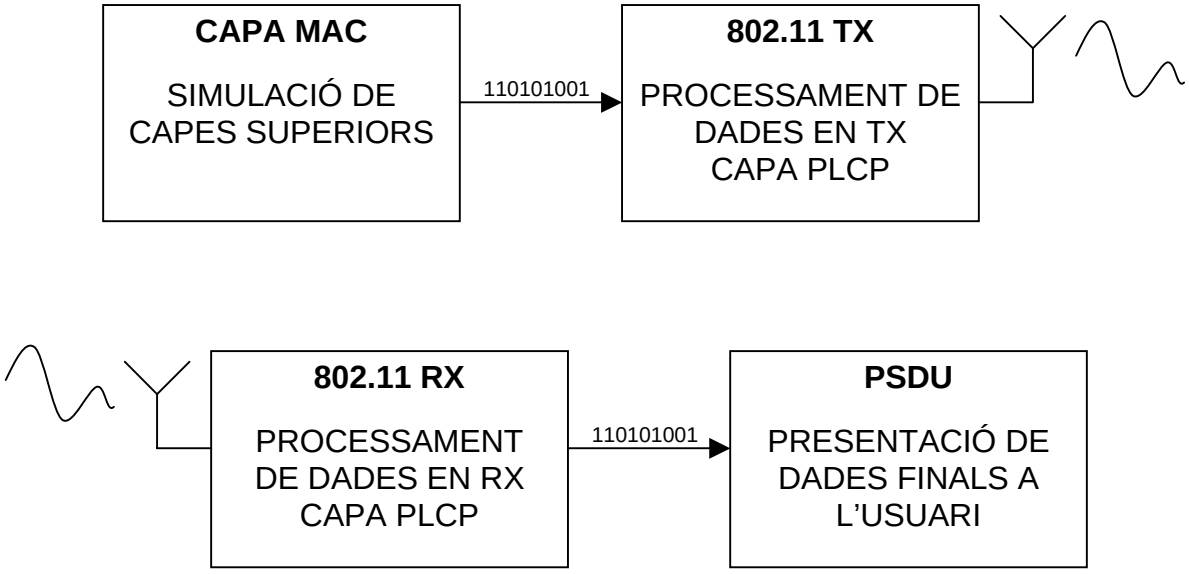


Fig. 4.1 Procés de transmissió

4.4 Funcionament general TX

La següent figura **(Fig. 4.1)** és un esquema que mostra la cadena d'emissió des que els bits arriben de capes superiors fins que finalment són entregats al medi.

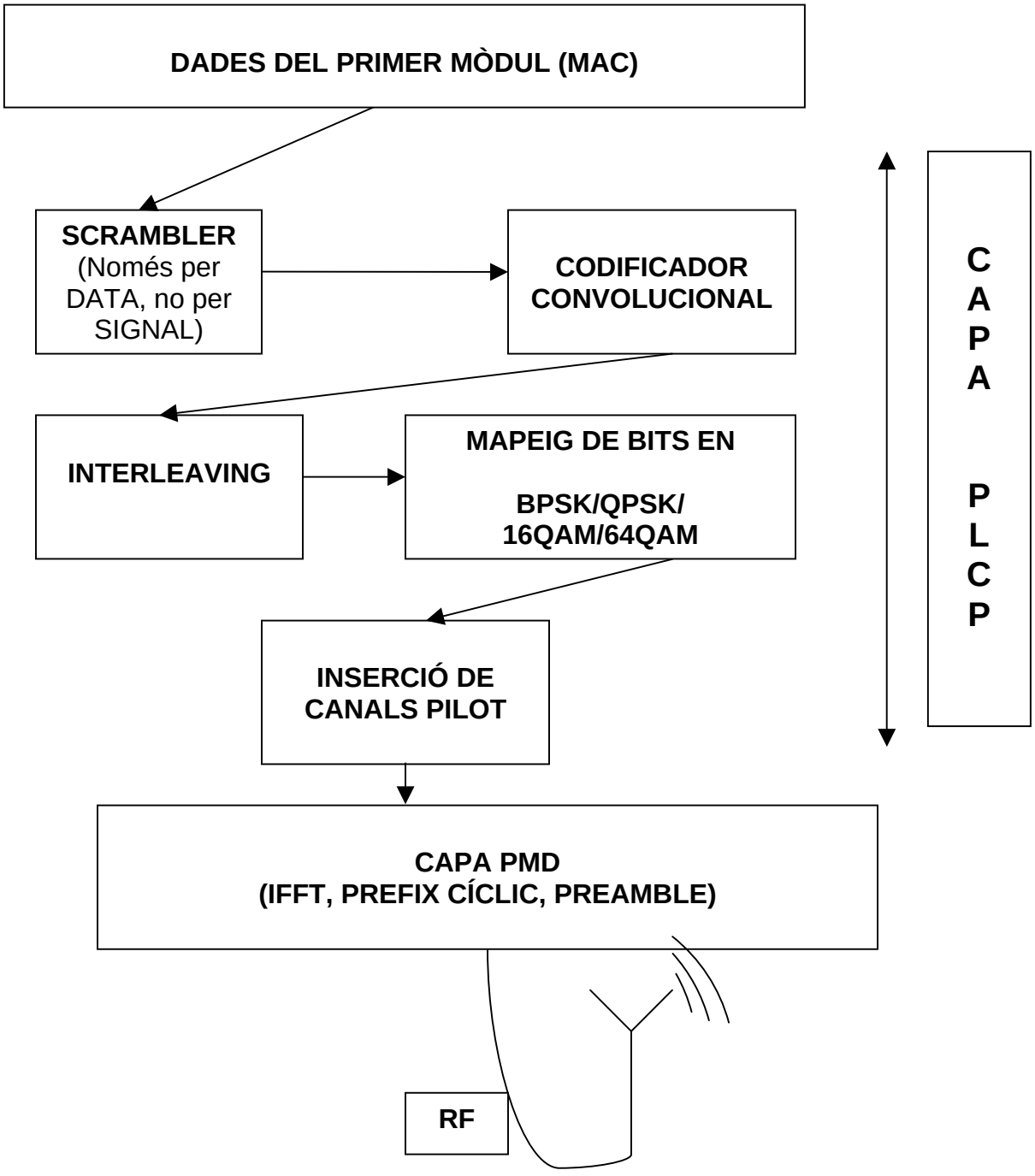


Fig. 4.1 Procés de transmissió

El mòdul encarregat del processament i enviament de dades (802.11 TX) és un bucle que s'executa mentre la capa MAC passi dades a la capa PHY (per les proves s'ha utilitzat un bucle infinit). Per cada bucle s'obté, si tot ha anat correctament, un paquet PPDU com el de la figura **(Fig. 4.2)**.

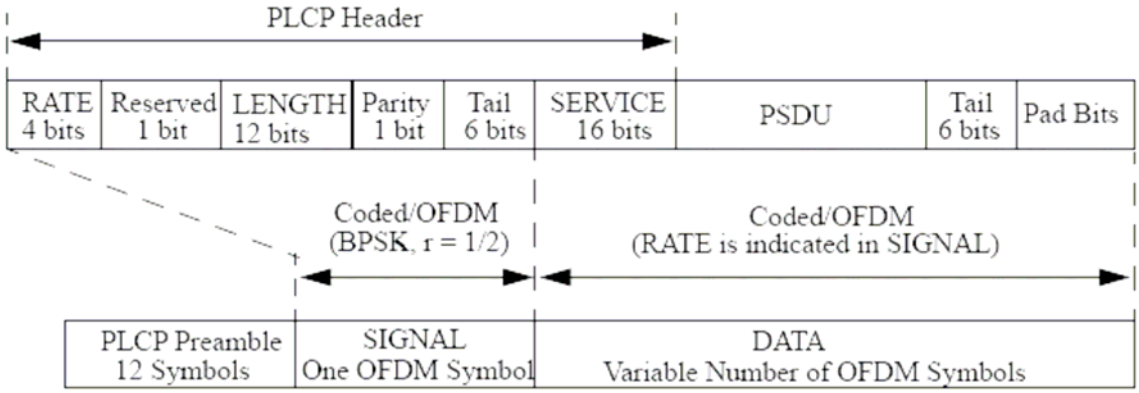


Fig. 4.2 Format frame PPDU

Tot i que realment es treballa a nivell de bit, degut a que un tipus bit no existeix a C, s'han utilitzat tipus char (1B) per representar 1 bit. Així doncs, per representar per exemple els 4 bits del camp RATE s'utilitza un vector de tipus char de 4 posicions.

A continuació es detallen dades necessàries per a la definició de vectors:

- Longitud RATE: 4 bits.
- Reserved : 1 bit.
- Parity : 1 bit.
- Tail SIGNAL: 6 bits (sempre a 0).
- SERVICE: 6 bits (sempre a 0).
- Longitud màxima PSDU binari: 111111111111 (12 bits), per tant 4095 octets, és a dir, bits que la capa MAC pot passar a la capa PHY en concepte de PSDU.
- Tail DATA: 6 bits (sempre a 0).
- PAD bits: # variable (sempre a 0).

Com es pot observar, no es necessari que la capa MAC proporcioni informació redundant, aquells bits que sempre es mapen a 0 no cal passar-los. Per tant, la capa MAC passarà a la capa PHY (mòdul 802.11 TX) els camps: RATE, Reserved, LENGTH i PSDU.

Totes aquestes dades sumen un total de com a màxim 4+1+12+4095 = 4.112 bits.

Per tant es necessita com a màxim guardar 4.112 bits procedent de la capa MAC per paquet PPDU.

4.4.1 Generació del camp SIGNAL

'msg' és una estructura de tipus Tppdu que conté les dades del camp SIGNAL i les del camp DATA.

headers.h

```
typedef struct
{
    char signal[24];
    char data[MAX_NUM_BITS_DATA];
}Tppdu; //Estructura on es guarden les dades necessàries per fer el missatge
```

El primer mòdul, el que simula la capa MAC, enviarà al segon, 802.11 TX, les dades necessàries per poder processar el paquet d'acord al format establert a l'apartat 4.3.1 (vector data_flow). Per això, s'utilitzaran les funcions de transferència de dades que facilita el CM.

Mòdul: capa MAC (vector amd dades de prova)

```
  long int time1, time0;
  char data_flow[77]={1,1,0,1,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0
```

Una vegada s'ha rebut aquest vector amb totes les dades, es poden omplir els 24 bits del camp SIGNAL a l'estructura msg.

Mòdul: 802.11 TX (Omplint l'estructura SIGNAL)

```
// ****************************  SIGNAL  **************

// OMPLIM ELS CAMPS DE SIGNAL

        msg.signal[0]=data_flow_101[0];//RATE
        msg.signal[1]=data_flow_101[1];
        msg.signal[2]=data_flow_101[2];
        msg.signal[3]=data_flow_101[3];
        msg.signal[4]=data_flow_101[4];//RESERVED
        msg.signal[5]=data_flow_101[5];//LENGTH
        msg.signal[6]=data_flow_101[6];
        msg.signal[7]=data_flow_101[7];
        msg.signal[8]=data_flow_101[8];
        msg.signal[9]=data_flow_101[9];
        msg.signal[10]=data_flow_101[10];
        msg.signal[11]=data_flow_101[11];
        msg.signal[12]=data_flow_101[12];
        msg.signal[13]=data_flow_101[13];
        msg.signal[14]=data_flow_101[14];
        msg.signal[15]=data_flow_101[15];
        msg.signal[16]=data_flow_101[16];
        suma=0;
        for(i=0;i<17;i++) //PARITY BIT 0-16
            suma+=msg.signal[i];
        if (suma%2==0)
            msg.signal[17]=0;
        else
            msg.signal[17]=1;
        msg.signal[18]=0;//TAIL BITS
        msg.signal[19]=0;
        msg.signal[20]=0;
        msg.signal[21]=0;
        msg.signal[22]=0;
        msg.signal[23]=0;
```

A continuació es detallen els passos a seguir per codificar el camp SIGNAL:

1. **Enviar el camp msg.SIGNAL al convolucionador**

```
// ENVIEM EL CAMP SIGNAL AL CONVOLUCIONADOR

        //Convolucionem el primer bit

    bits_in[0]=msg.signal[0];

        //Convolucionem tot el camp SIGNAL

        for(i=0;i<24;i++)
        {
            bits_out[0]=0;
            bits_out[1]=0;
            convol(bits_in, bits_out);
            bits_encoded[2*i]=bits_out[0];
            bits_encoded[2*i+1]=bits_out[1];

            for(j=5;j>=0;j--)
                bits_in[j+1]=bits_in[j];
            if(i!=23)
                bits_in[0]=msg.signal[i+1];
        }
```

La funció del convolucionador és: convol(.)

Com s'ha explicat a l'apartat 3.5.5 del codificador convolucional, per cada bit que entra, surten 2, (R=1/2), per tant, bits_in es va nutrint dels valors emmagatzemats a msg.SIGNAL i la funció retorna 2 bits (bits_out) resultat de la convolució.

Funció de convolució

```
void convol(char *in, char*out)
{

    char pol1[7]={1,0,1,1,0,1,1};
    char pol2[7]={1,1,1,1,0,0,1};
    int i;

    for(i=0; i<7; i++)
    {
        if (pol1[i]==1)
            out[0]+=in[i];
        if (pol2[i]==1)
            out[1]+=in[i];
    }
    out[0]%=2;
    out[1]%=2;

}
```

El vector bits_encoded emmagatzema el resultat de la convolució. Aquest vector té com a màxim 48 posicions, ja que entren 24 bits per convolucionar i amb un RATE R=1/2 surten el doble, 48.

2. Interleaving del camp SIGNAL

```
// INTERLEAVING DEL CAMP SIGNAL

        fperm(perm, 48, 1); //48 perquè són per 6 Mbps
        for(i=0; i<48; i++)
            bits_interl[perm[i][1]]=bits_encoded[i];
```

La funció que efectua l'Interleaving és: fperm(perm, N_{CBPS},N_{BPSC})

On:

'perm' és una matriu de [MAX_N_{CBPS}][2] posicions de integers.

Aquesta funció emmagatzema a la matriu perm, els índex de reordenació (interleaving) pel camp SIGNAL d'acord a l'explicat a l'apartat 3.5.4.

Funció d'interleaving

```
void fperm(perm_matrix perm, int cbps, int bpsc)
{
    int i,s, suma;

    for (i=0; i<cbps; i++)
    {
        perm[i][0]=(cbps/16)*(i%16)+(i/16);
        if(bpsc/2>1)
            s=bpsc/2;
        else
            s=1;
        suma=(perm[i][0]+cbps-(16*perm[i][0]/cbps))%s;

        perm[i][1]=s*(perm[i][0]/s)+suma;

    }
}
```

El vector bits_interl guarda els bits ja reordenats. Aquest vector tipus char té 48 posicions, exactament les mateixes que resulten de la convolució, ja que simplement es realitza una reorganització en la posició dels valors.

3. Mapeig i canals pilot

Degut a que el camp SIGNAL sempre té una RATE de 6 Mbps s'escull la modulació digital BPSK (Veure 3.5.6) . El vector bit_signal és de tipus char amb 128 posicions (64 punts de la IFFT x 2, I+jQ) que guarda els símbols finals a transmetre a la capa PMD, la qual farà la IFFT i la resta del procés de transmissió.

```
// MAPEIG DELS BITS de SIGNAL EN BPSK

        for(i=0;i<6;i++)
        {
            bit_signal[2*i]=0;
            bit_signal[2*i+1]=0;
        }
        for(i=6,j=0;i<59;i++)
        {
            if (i==11 || i==25 ||i==39 ||i==53) //PORTADORES PILOT
            {
                bit_signal[2*i]=1*pilot[sim_act];
                bit_signal[2*i+1]=0;
                bit_signal[2*i]=1*pilot[sim_act];
                bit_signal[2*i+1]=0;
                bit_signal[2*i]=1*pilot[sim_act];
                bit_signal[2*i+1]=0;
                bit_signal[2*i]=-1*pilot[sim_act];
                bit_signal[2*i+1]=0;
            }

            else if (i==32) //Portadora 0
            {
                bit_signal[2*i]=0;
                bit_signal[2*i+1]=0;
            }
            else
            {
                if (bits_interl[j]==0)
                    bit_signal[2*i]=-1;
                else
                    bit_signal[2*i]=1;
```

4.4.2 Generació del camp DATA

El procés per generar el camp DATA és força similar al de SIGNAL.

Tal i com s'ha dit a l'apartat anterior al camp msg.data s'emmagatzemen els camps SERVICE+PSDU+TAIL+PAD. Per poder realitzar aquesta operació cal haver llegit dades del camp SIGNAL, concretament els camps RATE i LENGHT ja que indicaran el tamany del PSDU i els de bits de Tail d'acord la configuració de velocitat seleccionada. A partir d'aquestes dades s'aconsegueixen cinc valors imprescindibles per continuar amb la codificació (RATE, condingrate, N_{bpsc}, N_{cbps} i N_{dbps}) (Veure 3.1.2).

Inicialitzant valors

```
    dbps=24;
}
if (msg.signal[0]==1 && msg.signal[1]==1 && msg.signal[2]==1 && msg.signal[3]==1)
{
    rate=9;
    codingrate=3;
    bpsc=1;
    cbps=48;
    dbps=36;
}
if (msg.signal[0]==0 && msg.signal[1]==1 && msg.signal[2]==0 && msg.signal[3]==1)
{
    rate=12;
```

Ara que ja es tenen les dades necessàries, només cal emmagatzemar al camp msg.data els següents valors en aquest mateix ordre (Veure **Fig. 4.2** per més informació):

SERVICE + PSDU + TAIL + PAD

Per incloure els bits de Pad cal calcular primer la longitud final del missatge (sense Padding) i afegir els bits necessaris per adaptar el missatge segons l'explicat a l'apartat 3.4.3 referent al Padding.

A continuació es detallen els passos a seguir per formar el camp DATA:

1. **Scrambler**

La funció scrambler(), s'encarregarà de realitzar l'encriptació de bits d'acord a l'apartat 3.5.1. Degut a que el camp DATA té una longitud variable, no es pot especificar en els següents apartats el tamany de cada vector tal i com s'ha fet per SIGNAL. Ja que el disseny de l'aplicació "MyWlan" utilitza memòria estàtica es treballarà amb el vector sencer tot i que aquest no quedi omplert.
La funció d'Scrambler requereix restaurar a posteriori els bits de Tail i Pad a 0's.

Funció scrambler

```
void scrambler(int ndata, char *bits_scrambled, char *data)
{
    char sequence[127];
    int result,i,j;
    char seed[7]={1,1,1,1,1,1,1};



    for (i=0; i<127;i++)
    {
        if (seed[0]==seed[3])
            result=0;
        else
            result=1;
        for(j=0;j<6;j++)
            seed[j]=seed[j+1];
        seed[6]=result;
        sequence[i]=result;
    }

        for(i=0,j=0; i<ndata; i++,j++)
    {
        if(j==127)
            j=0;
        if(sequence[j]==data[i])
            bits_scrambled[i]=0;
        else
            bits_scrambled[i]=1;
    }

}
```

2. Codificador Convolucional

La funció convol(.) és exàctament la mateixa que l'utilitzada en el camp SIGNAL. No obstant degut a que el camp DATA no sempre té una taxa R=1/2, cal aplicar puncturing en els casos necessaris.

Funció de puncturing per R=2/3

```
if (codingrate==2) //RATE 2/3
{

        for(i=0;i<ndata;i++)
    {
        bits_out[0]=0;
        bits_out[1]=0;
        convol(bits_in, bits_out);
        data_convol[2*i]=bits_out[0];
        data_convol[2*i+1]=bits_out[1];


        for(j=5;j>=0;j--)
            bits_in[j+1]=bits_in[j];
        if(i!=ndata-1)
            bits_in[0]=bits_scrambled[i+1];
    }
        for(z=0, a=0; z<2*ndata; z++,a++)
        {
            if (a==3)
            {
                for(b=z; b<2*ndata; b++)
                    data_convol[b]=data_convol[b+1];
                a=0;
            }
        }

}
```

Una vegada fet puncturing el camp DATA convolucionat es guarda al vector data_convol.

3. Interleaving

De nou, la funció fperm(.), abans utilitzada pel camp SIGNAL, realitza l'operativa d'interleaving.

Funció Interleaving

```
// INTERLEAVING DEL CAMP DATA

        fperm(perm, cbps, bpsc);
        sim_act=1;
        while((nsymb-sim_act)>=0)
        {
            for(i=0; i<cbps; i++)
            data_interl[(sim_act-1)*cbps+perm[i][1]]=data_convol[(sim_act-1)*cbps+i]
            sim_act++;
        }
```

El vector data_interl guarda les dades de data_convol amb l'interleaving fet.

4. Mapeig i canals pilot

Segons el RATE a aplicar, es guardarà al vector bit_data els bits finals ja mapats en una BPSK, QPSK, 16-QAM o 64-QAM segons correspongui.

Funció mapeig de bits

```
if(rate==24 || rate==36) //Cas per una 16-QAM
    {
    for(i=0;i<6;i++)
        {
            bit_data[2*i]=0;
            bit_data[2*i+1]=0;
        }
        for(i=6, j=0;i<59;i++)
        {
            if (i==11 || i==25 ||i==39 ||i==53)
            {
                if (sim_pilot==127)
                    sim_pilot=0;

                if (i==11)
                {
                bit_data[2*i]=1*pilot[sim_pilot];
                bit_data[2*i+1]=0;
                }
                if (i==25)
                {
                bit_data[2*i]=1*pilot[sim_pilot];
                bit_data[2*i+1]=0;
                }
                if (i==39)
                {
                bit_data[2*i]=1*pilot[sim_pilot];
                bit_data[2*i+1]=0;
```

4.5 Funcionament general RX

Pel cas de la recepció de paquets, la següent figura **(Fig. 4.3)** mostra els diferents estats pels que passen els bits, des que entren per RF fins que s'obtenen les dades enviades originalment.

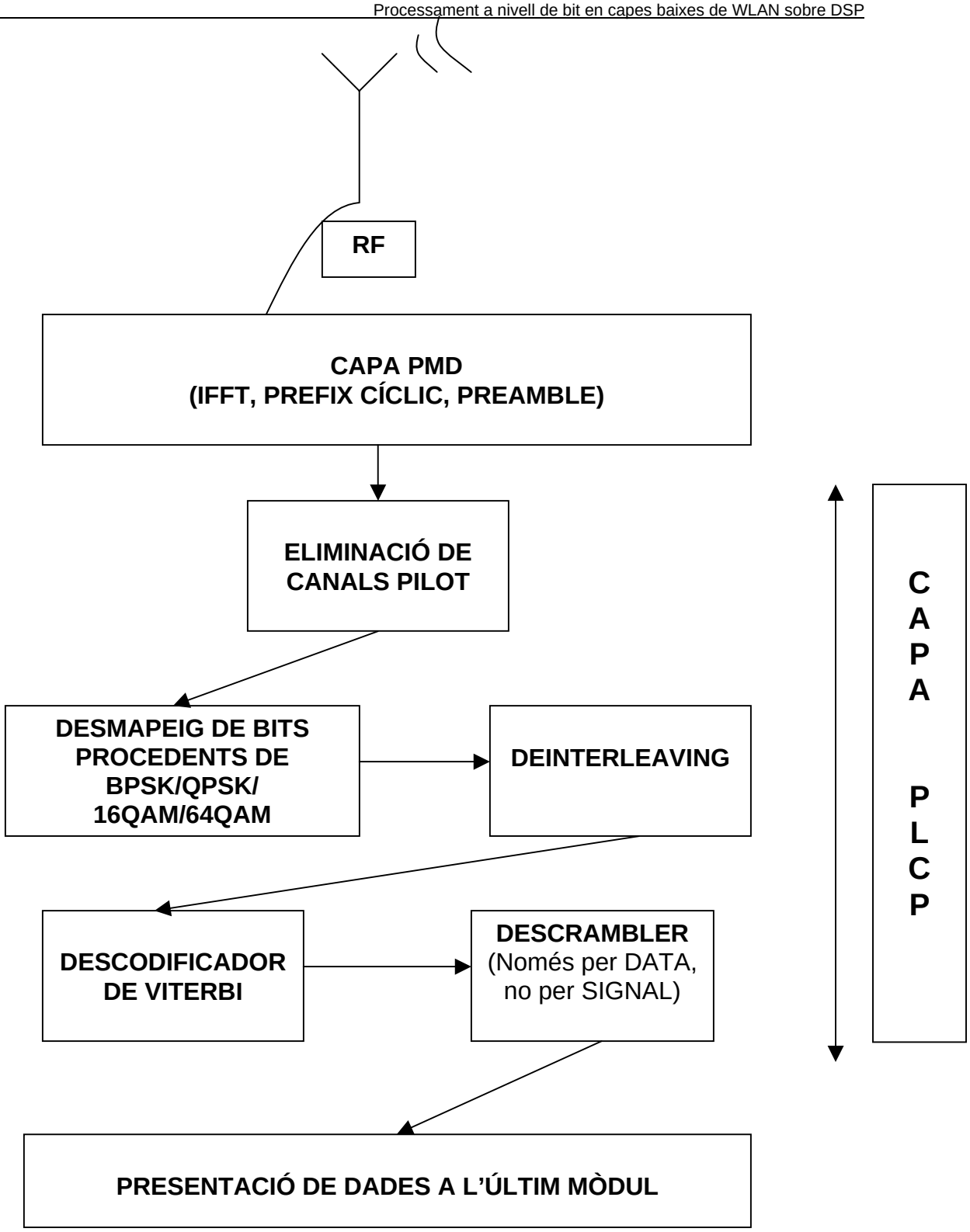


Fig. 4.3 Procés de recepció

El mòdul encarregat de la recepció i descodificació de dades (802.11 RX) és un bucle infinit que llegirà contínuament del canal les dades entrants a l'espera de rebre símbols. Cal tenir en compte que a la figura es pot veure tot el procés de recepció, però es torna a recordar que aquest codi implementa la totalitat de la capa PLCP i no així la PMD.

El primer símbol que arriba al receptor quan s'aconsegueix fer la lectura correctament d'un paquet és el SIGNAL seguit de tants símbols DATA com siguin necessaris. En cas de que per error de lectura no es pugui recuperar el símbol SIGNAL, tot i que la resta de símbols DATA arribin perfectament seran descartats ja que falta informació imprescindible per la recuperació d'aquests.

4.5.1 Descodificació camp SIGNAL

Ja que el camp SIGNAL sempre va codificat amb BPSK a 6 Mbps, es guarden els valors rebuts (-1 / 1) al vector bits_rebuts, d'una longitud de 128 posicions (64 bits corresponents als punts de la IFFTx2, I+jQ).

Lectura dels bits_rebuts pel camp SIGNAL

```
n=ReadFlow(flw[0],128,bits_rebuts);
res=-1;
```

A continuació es detallen els passos a seguir per recuperar el camp SIGNAL:

1. **Desmapeig**

   Els valors rebuts inicialment (-1 / 1) es desmapen d'acord a la taula de mapeig d'una BPSK (Veure 3.5.6) i es guarden els valors al vector bits_interl, d'una longitud de 48 posicions (48b).

   Funció desmapeig símbol SIGNAL

```
// Desmapegem els bits de la BPSK.


    for(i=0, j=0; j<NUM_BITS_SIMBOL; j++)
    {
        if(j%2==0 && j>11 && j<118 && j!=22 && j!=50 && j!=64 && j!=78 && j!=106)
        {
            if(bits_rebuts[j]<=0)
                bits_interl[i]=0;
            else
                bits_interl[i]=bits_rebuts[j];
            i++;
        }
    }

    //Tenim a bits_interl 48 bits interleaved
```

   NOTA: Les portadores que contenien els canals pilot deixen de ser útils una vegada s'ha fet l'estimació del canal, per tant les rebutjem en aquest primer pas.

2. Deinterleaving

La funció per desfer l'Interleaving és: fperm(.)
Aquesta funció emmagatzema a la matriu perm, els índex de reordenació (interleaving) pel camp SIGNAL.

Funció fperm()

```
void fperm(perm_matrix perm, int cbps, int bpsc)
{
    int i,s;

    if(bpsc/2>1)
        s=bpsc/2;
    else
        s=1;


    for (i=0; i<cbps; i++)
    {

        perm[i][0]=s*(i/s)+(i+(16*i/cbps))%s;

        perm[i][1]=16*perm[i][0]-(cbps-1)*(16*perm[i][0]/cbps);

    }
}
```

Funció Deinterleaving

```
//Desfem l'interleaving


    fperm(perm, 48, 1); //48 perquè són per 6 Mbps, n=NCBPS=48
    for(i=0; i<48; i++)
        bits_deinterl[perm[i][1]]=bits_interl[i];

    //A bits _deinterl tenim els 48 bits deinterleaved
```

El vector bits_deinterl guarda els bits ja reordenats. Aquest vector és un tipus char de 48 posicions.

3. Descodificador de Viterbi

La funció viter(.) realitza la descodificació de Viterbi.

Descodificació de Viterbi

```
//Decodificador de VITERBI

    viter(bits_deinterl, 18, 1, resultat);
```

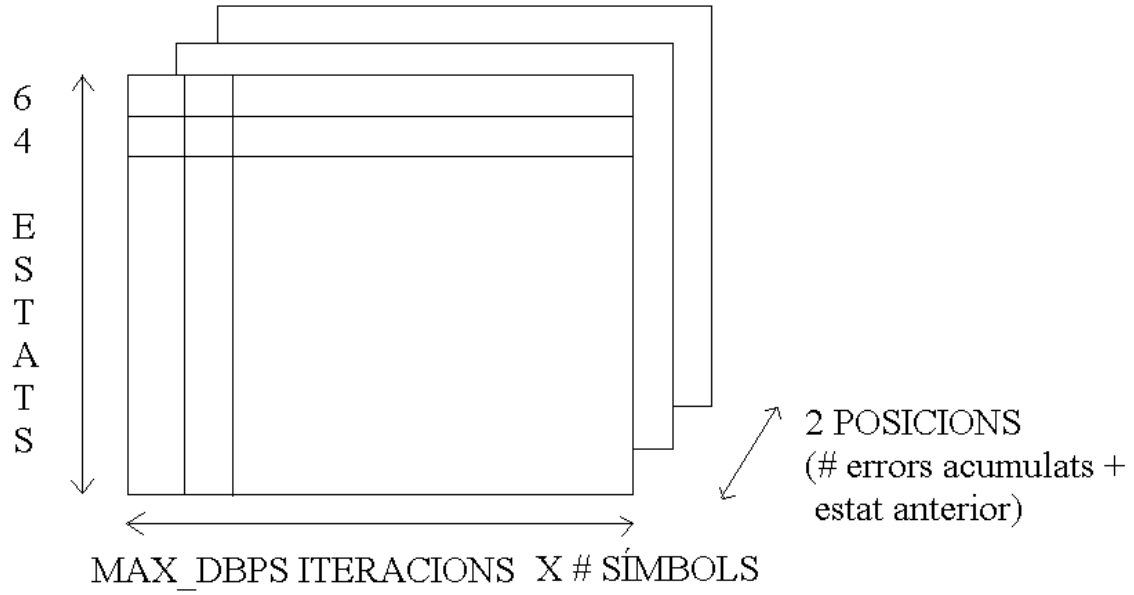
El descodificador de Viterbi és un dels processos més complexos del programa ja que són moltes les operacions, tal i com s'ha explicat al capítol 3 que s'han de fer. Per això, es presenta a continuació una explicació acurada de les funcions implementades dins la descodificació de Viterbi.

1. Inicialització de la matriu ***sit_estat***

La matriu sit_estat s'encarrega de mantenir un registre de tots els possibles camins que els bits rebuts poden seguir d'acord a la codificació convolucional practicada en transmissió. Gràcies a aquesta matriu, al final de la descodificació, es podrà seleccionar el camí òptim i fent marxa enrere recuperar els bits enviats "suposadament" en TX.

Aquesta matriu queda definida així:

```
int sit_estat[MAX_DBPS][64][2];
```



D'acord a l'explicat al capítol 3 a l'apartat del codificador convolucional (3.5.2), existeixen 64 possibles estats, ja que la memòria del codificador és 7 i per tant, 2^(7-1)= 64 estats.
Pel que fa al número d'iteracions es recorda que N_{DBPS} fa referència al número de bits de dades útils per símbol OFDM. Pel cas SIGNAL es sap que sempre són 24 bits útils generats i per tant el màxim número d'iteracions seran 24 per aconseguir recuperar la totalitat de bits que van ser convolucionats. Però, pel camp DATA es

poden arribar a tenir fins a 216 bits útils per símbol OFDM. A més, cal tenir en compte que la descodificació de Viterbi es realitza en una sola vegada per a la totalitat de símbols OFDM rebuts per paquet PPDU ja que aquests van estar codificats de manera annexada.
Finalment existeixen dues posicions tal i com es pot veure a la matriu, en les quals es guarden l'últim estat del que es procedeix i el número d'errors comesos fins el moment per aquell camí.

2. Generació de tots els estats possibles

Per tal de poder desenvolupar la descodificació de Viterbi, cal conéixer a priori quins seran els possibles 64 estats pels quals els bits es poden moure. Per tant, l'únic que s'ha de fer és omplir una matriu ***estat [ ][ ]***, amb la combinació dels 64 estats, és a dir:

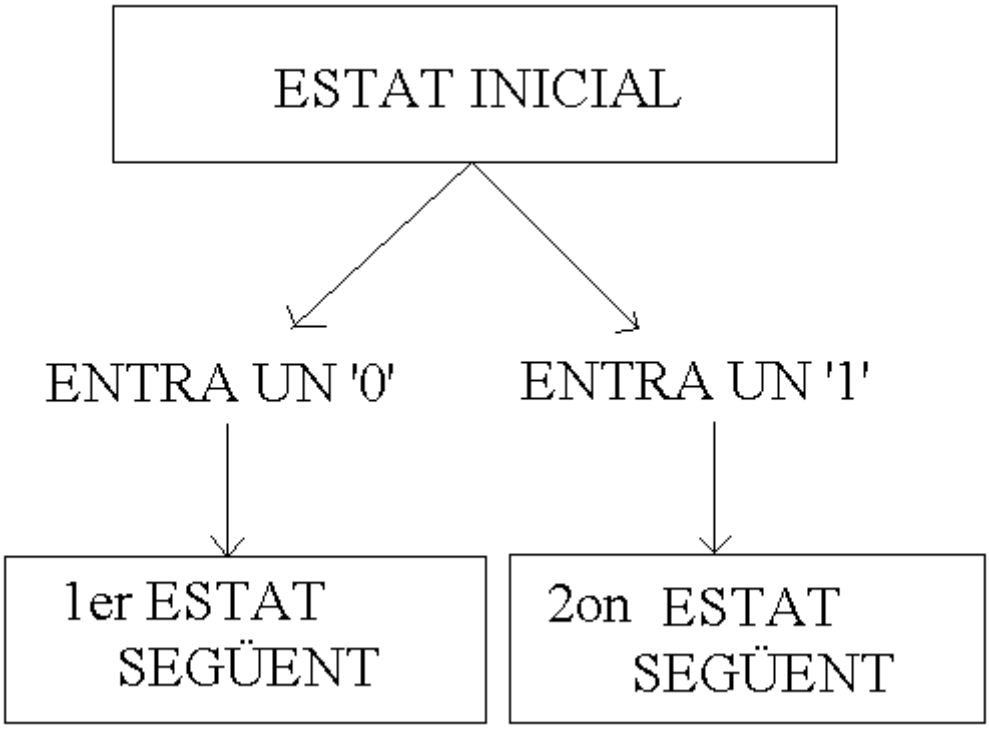
000000
000001
000010
....
111111

La funció que realitza aquesta operació és:

```
//Generació de tots els estats possibles

    for(col=0; col<6; col++)
    {
        n=1;

        for(cont=0; cont<(6-col); cont++)
            n*=2;


        for(i=0; i<64; i+=n)
        {
            for(j=i; j<i+(n/2); j++)
                estat[j][col]=0;

            for(z=j; z<j+(n/2); z++)
                estat[z][col]=1;
        }
    }
```

3. Diagrama de Trellis

Tot i tenir generats tots els possibles estats cal dir que no és possible passar d'un estat a un altre qualsevol, sinó que d'acord a l'esquema del codificador convolucional **(Fig. 3.9)** hi haurà una sèrie de canvis d'estats permesos i d'altres que no. Això en resum vol dir haver de generar el diagrama de Trellis.
La funció que realitza aquesta operació segueix el següent esquema:



Per tant, la funció agafa un estat (ha de comprovar tots un per un) i busca a quins dos estats futurs és pot passar, depenent si l'entrada és un '1' o bé, un '0'.
Els resultats d'aquesta verificació es guarden a la matriu anomenada camins[i][j].
On:

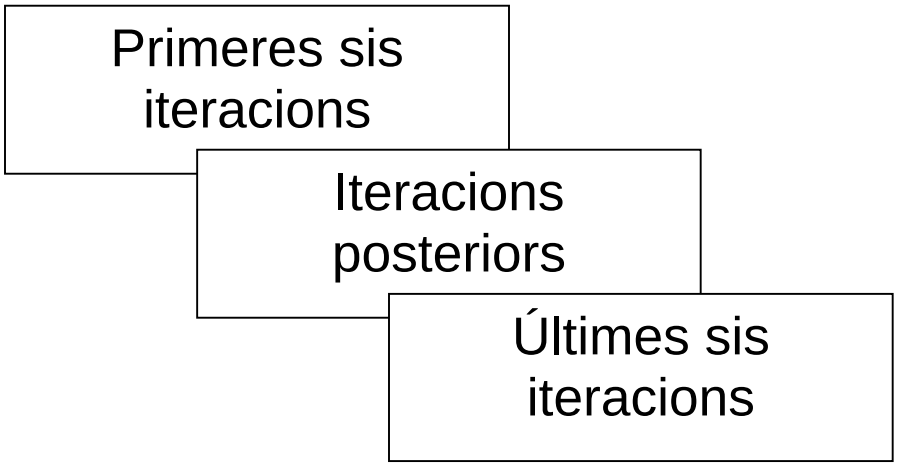
i-> número de coincidència. Aquí cada fila correspon a un possible combinació vàlida de (estat_anterior -> estat següent). Hi ha 128 possibles combinacions, ja que hi ha 64 estats a dos estats futurs per cadascún d'ells.

j-> format per cinc columnes que guarden respectivament:

- col 0 -> Estat inicial
- col 1 -> Estat següent
- col 2 -> Output A (el primer bit que surt quan s'efectua aquest canvi d'estat)
- col 3 -> Output B (el segon bit que surt quan s'efectua aquest canvi d'estat)
- col 4 -> Input (bit que va entrar al codificador per produir-se aquest canvi)

4. Procés de descodificació

Una vegada ja ha quedat tot inicialitzat d'acord els tres últims apartats comença el procés de descodificació per tal d'aconseguir recuperar els bits enviats.
Està dividit en tres parts:



Aquesta divisió està feta perquè per a les sis primeres iteracions no hi ha dos possibles estats futurs, dues branques per estat, sinó que només hi ha un. I perquè per les últimes sis, ja que hi ha bits de Tail, ens hem d'assegurar d'acabar a l'estat 0.

Tenint en compte l'anteriorment dit, la funció Viterbi de manera iterativa anirà buscant per cada estat, de qual pot haver vingut decidint segons els errors acumulats pel camí seleccionat aquell pel que hagi acumulat menys.
Totes aquestes dades es van guardant a la matriu ja abans comentada, sit_estat [ ] [ ] [ ].

Quan s'hagi finalitzat tot el cicle, a l'última iteració es tindrà el camí final decidit i només caldrà fer marxa enrere i anar guardant al vector resultat [ ], els bits ja descodificats amb l'ajut de la matriu camins [ ] [ ].

4. Omplir l'estructura msg

Finalment, amb els bits descodificats per l'algorisme de Viterbi, s'omple l'estructura msg, en aquest cas el camp msg.SIGNAL, amb els bits finals recuperats d'aquest primer símbol.

```
//OMPLIM EL CAMP MSG.SIGNAL

    for(i=0; i<24; i++)
{
        msg.signal[i]=resultat[i];
        fprintf(fp, "%d ", msg.signal[i]);
}
```

4.5.2 Descodificació del camp DATA

Una vegada feta la lectura del símbol SIGNAL del paquet PPDU, s'haurien de rebre els símbols corresponents al camp DATA. En cap cas si el símbol SIGNAL es perd es podran llegir els símbols DATA perquè el símbol perdut conté informació imprescindible per realitzar la lectura. En aquesta situació tots els símbols DATA serien descartats pel programa.

1. **Desmapeig**

   Una vegada obtinguda la informació referent als símbols DATA facilitada per la lectura del símbol SIGNAL cal fer el desmapeig de la totalitat dels símbols DATA que composen aquest paquet PPDU, obtenint N_{CBPS} bits per símbol DATA.

   Interpretació camp SIGNAL

```
if (msg.signal[0]==1 && msg.signal[1]==1 && msg.signal[2]==0 && msg.signal[3]==1)
{
    rate=6;
    codingrate=1;
    bpsc=1;
    cbps=48;
    dbps=24;
```

   El procés de desmapeig és igual que pel camp SIGNAL només que pels símbols DATA a més dels 6 Mbps hi ha altres velocitats i per tant altres modulacions (veure **Taula 3.1**).

```
  if(rate==48 || rate==54) //Cas per una 64-QAM
  {
    for(i=0, j=0; j<NUM_BITS_SIMBOL; j+=2)
    {
      if(j>11 && j<118 && j!=22 && j!=50 && j!=64 && j!=78 && j!=106)
      {
          //FASE

          if(data_rebuts[j+(sim_act-1)*128]<=-6*1000/sqrt(42))
          {
              data_interl[i+(sim_act-1)*cbps]=0;
              data_interl[i+1+(sim_act-1)*cbps]=0;
              data_interl[i+2+(sim_act-1)*cbps]=0;
          }
          if(data_rebuts[j+(sim_act-1)*128]>-6*1000/sqrt(42) && data_rebuts[
```

El resultat del desmapeig de tots els símbols DATA que composen el PPDU s'emmagatzema al vector data_interl [ ] que tot i que la seva capacitat màxima està definida, les posicions a omplir correspondran a N_{CBPS} x número de símbols OFDM que conformen el paquet PPDU.

2. Deinterleaving

De nou la funció per desfer l'Interleaving és: fperm(.).

El resultat es guarda al vector data_deinterl [ ]. El tamany d'aquest vector continua essent el mateix que el de data_interl [ ], per ser únicament una reordenació dels valors.

Només si s'ha utilitzat la tècnica de puncturing a la codificació, es desfarà en aquest moment. Els bits resultants queden emmagatzemats al vector data_depunct. En cas de no haver utilitzat puncturing (perquè la RATE seleccionada comportés una codificació convolucional R=1/2) es fa una còpia de les dades de data_deinterl a data_depunct. Recordar que en cas d'haver utilitzat puncturing cal reinsertar els bits que es van eliminar en emissió amb 0's.

```
//Desfer Puncturing

    if (codingrate==1)
    {
        for(z=0; z<nsymb*cbps; z++)
            data_depunct[z]=data_deinterl[z];
    }
```

3. Descodificador de Viterbi

Per descodificar el camp DATA s'apliquen les mateixes funcions i criteri que pel camp SIGNAL. Únicament dir que el procés serà més llarg perquè poden haver molts símbols per descodificar i s'han de fer tots seguits, ja que l'algorisme de Viterbi no permet decidir quins símbols foren enviats fins que no s'ha arribat al final del procés, en aquest cas, a les últimes sis iteracions amb els bits de Tail.

En aquest cas, el vector que guarda els bits descodificats per l'algorisme de Viterbi és data_deconvol [ ] que té un tamany igual a la meitat de data_depunct. (Recordem que el codificador convolucional treia 2 bits per cada 1 d'entrada, R=1/2).

4. Desfer l'Scrambling

Degut a que el camp DATA passa per un scrambler en el procés de codificació, és el moment de desfer-lo, utilitzant la funció scrambler(.), la mateixa que es va utilitzar a la codificació (la funció és síncrona). Els bits queden guardats al vector bits_descrambled [ ] (de la mateixa longitud que data_deconvol [ ]) i són directament els originals enviats si tot ha funcionat correctament, i que per tant, passen a l'últim mòdul del sistema (PSDU) on es presenten les dades a l'usuari.

Crida a la funció Scrambler

```
scrambler(longit, bits_descrambled, data_deconvol);
```

Enviament de dades finals al mòdul PSDU

```
if (GetFlowStatus(flw[1])>0)
    n=WriteFlow(flw[1],lon_psdu,msg.data);
```

CAPÍTOL 5. Conclusions

Com s'ha pogut veure en els dos capítols anteriors, són moltes les operacions per les que passen els bits des que capes superiors entreguen els bits a la subcapa PLCP i aquesta les entrega a la següent subcapa de la capa física, PMD.

Amb això volem dir que un únic error, un únic bit mal processat canvia per complet el resultat final. Com és evident, és "impossible" construir un programa d'aquestes dimensions sense errors a la primera. Però una vegada finalitzat estem en condicions d'assegurar que el codi és correcte i no hi ha cap cas en que s'estiguin fent operacions il·legals per l'ample banc de proves utilitzat. A l'annex es poden trobar moltes d'aquestes proves, mostrant els estats pel que passen els bits inicials fins arribar al final del procés.

La realització del projecte es va trobar amb dos obstacles, per un costat un Sistema Operatiu un tant desconegut personalment, Linux, i per una altra banda, una nova eina, el Comunications Manager, totalment per explorar.

Una vegada instal·lat Linux i el CM ens trobem amb els primers problemes. Linux no permet l'apertura de ports i a més, no permet al CM escriure registres per temes de permissos. Tot i això, aquests inconvenients s'aconsegueixen solucionar poc temps després.

Degut a que fer el debugging sota Linux i el CM era força enfarragós, una primera versió del programa es realitza sota Windows amb l'ajut del Visual Studio. Això ens ajuda molt a detectar ràpidament els errors i poder-los solucionar. Comentar que l'apartat que realitza el puncturing i la descodificació de Viterbi, van ser els punts més complicats.

Tot i que el programa funciona correctament, cal fixar-se en els temps en que s'executa l'operació de codificació i descodificació del frame final. L'annex mostra en detall els temps utilitzats pel programa en l'enviament i recepció de dades i quant difereixen dels exigits per les especificacions.

Com a conclusió final i personal dir que fins que no vaig llegir les especificacions de la norma 802.11, no m'imaginava la quantitat de passos que calen per poder enviar un mail o entrar a una web des d'un ordinador connectat a una xarxa sense fils. No obstant, és molt gratificant veure com els bits que obtens amb el teu programa coincideixen amb els de l'exemple de les especificacions.

Bibliografia

[1] Huidobro, J. “Wi-Fi”. http://www.monografias.com/trabajos14/wi-fi/wi-fi.shtml. [Consultada març de 2006]

[2] “IEEE”. http://es.wikipedia.org/wiki/ITU. [Consultada en marc de 2006]

[3] Gast, Matthew. *802.11 Wireless Networks: The Definitive Guide*, O’Reilly Publisher, Abril 2002.

[4] “CRM-03”. http://europa.eu.int/information_society/topics/telecoms/radiospec/doc/pdf/wrc_03_documents/com_2003_183_es.pdf. [Consultada març de 2006]

[5] P.Martí, *Algorismes de detecció i recuperació de referència per a modulacions transformades*, PhD Thesis, UPC, 2001.

[6] R.Van Nee, R. Prasad, *OFDM for Wireless Multimedia Communications*, Artec House Publishers, 2000.

[7] J. Heiskala, J. Terry, *OFDM Wireless LANs: a theoretical and practical guide*, Sams Publishing, 2002.

[8] S. B. Weinstein, P. M. Ebert, *Data Transmission by Frequency Division Multiplexing Using the Discrete Fourier Transform*, IEEE Trans. Comm., COM-19:628–634, Oct. 1971.

[9] Viterbi Algorithm. http://en.wikipedia.org/wiki/Viterbi_algorithm. [Consultada abril de 2006]

[10] DSP vs. Pentium http://www.valdesystems.com/media/VS1701_brochure.pdf. [Consultada abril de 2006]

[11] Communications Manager Tutorial [Release date: April 30th 2004, Document version: 1.0, CM version: 2.1.1, Biblioteca EPSC]

ANNEX A

Aquest annex té com a objectiu verificar el correcte funcionament del codi programat. Per aconseguir-ho, aquest capítol està dividit en quatre seccions.

L'apartat A.1 és un volcat de pantalles comentades indicant el procés per executar el CM junt amb l'aplicació *"MyWlan"*.
L'apartat A.2 és un recull de pantalles que mostren el funcionament del programa.
L'apartat A.3 conté el procés sencer d'enviament i recepció d'un paquet de prova.
L'apartat A.4 és un recull de resultats i conclusions.

A.1 Preparació de l'escenari on s'executarà l'aplicació

Abans de poder executar l'aplicació "MyWlan" és necessari que la plataforma CM estigui compilada al PC de testeig.
Per això, una vegada descomprimida l'eina CM a una carpeta qualsevol de l'equip amb drets d'administrador, obrim una Terminal i fem el següent:

Terminal
[root@localhost CM]# make

Amb això es creen a la carpeta /bin els binaris que conformen el CM i que més endavant utilitzarem per executar la nostra aplicació. Cal dir que com que aquesta aplicació ja estava programada i testejada, no ha de sortir cap tipus d'error durant la instal·lació.

Instal·lació Communications Manager

```
[root@localhost projecte20]# make
gcc -c -Wunused -O cm_time.c -o ./obj/cm_time.o
gcc -Wunused -Wshadow -O cdaemon.c ./obj/cm_time.o -o ./bin/cdaemon
gcc -Wunused -Wshadow -O3 bridge.c ./obj/cm_time.o -o ./bin/bridge
gcc -Wunused -Wshadow -O iconsole.c ./obj/cm_time.o -o ./bin/iconsole
gcc -c -Wunused -Wshadow -O2 shell.c -o ./obj/shell.o
gcc -c -Wunused -Wshadow -O2 shell_str.c -o ./obj/shell_str.o
gcc -c -Wunused -Wshadow -O2 shell_net.c -o ./obj/shell_net.o
gcc -O2 -c parser.c -o ./obj/parser.o
gcc ./obj/shell.o ./obj/shell_str.o ./obj/shell_net.o ./obj/parser.o ./obj/cm_ti
me.o -o ./bin/shell
gcc -c -Wunused -Wshadow -O2 runtb.c -o ./obj/runtb.o
gcc ./obj/runtb.o ./obj/shell_net.o  ./obj/cm_time.o -o ./bin/runtbd
gcc -c -Wunused -Wshadow -O3 statsd.c -o ./obj/statsd.o
gcc -c -Wunused -Wshadow -O3 stats_str.c -o ./obj/stats_str.o
gcc -c -Wunused -Wshadow -O3 stats_com.c -o ./obj/stats_com.o
gcc ./obj/statsd.o ./obj/stats_str.o ./obj/parser.o ./obj/cm_time.o ./obj/stats_
com.o -o ./bin/statsd
gcc -c -Wunused -O cm_api_str.c -o ./obj/cm_api_str.o
gcc -c  -Wunused -O cm_api.c -o ./obj/cm_api.o
ar cru ./lib/libcm_api.a ./obj/cm_api.o ./obj/cm_api_str.o ./obj/parser.o ./obj/
cm_time.o ./obj/stats_com.o
[root@localhost projecte20]#
```

Com s'ha explicat a la memòria, l'aplicació *"MyWlan"* consta de quatre mòduls.

Ara és el moment de compilar els quatre aplicatius: Capa MAC, 802.11 TX, 802.11 RX i PSDU. Tot i que nosaltres els hem donat noms descriptius, el CM requereix que aquests mòduls estiguin identificats mitjançant una numeració específica. En el nostre cas hem numerat els quatre mòduls del 101 al 104 respectivament.

Els mòduls es troben a la carpeta /modules del CM i per compilar-los cal obrir una terminal i executar:

Terminal
[root@localhost modules]# make

Si el codi no fa cap operació il·legal, el resultat del compilador ha de ser positiu.

Instal·lació dels mòduls *"MyWlan"*

```
[root@localhost projecte20]# cd modules
[root@localhost modules]# ls
altres_funcions2.c*  deinterleaving.c*  Makefile~*     modul_104.c*
altres_funcions.c*   headers2.h*        modul_101.c*   plcp_preamble.c*
bin/                 headers2.h~*       modul_101.c~*  scrambler2.c*
cm_ifc.h*            headers.h*         modul_102.c*   scrambler.c*
convol2.c*           headers.h~*        modul_102.c~*  viterbi.c*
convol.c*            interleaving.c*    modul_103.c*   viterbi.c~*
Debug/               Makefile*          modul_103.c~*
[root@localhost modules]# make
gcc -o ./bin/mod101 modul_101.c -L../lib -lcm_api
gcc -o ./bin/mod102 modul_102.c scrambler.c interleaving.c convol.c altres_funci
ons.c -L../lib -lcm_api
gcc -o ./bin/mod103 modul_103.c altres_funcions2.c convol2.c deinterleaving.c sc
rambler2.c viterbi.c -L../lib -lcm_api
gcc -o ./bin/mod104 modul_104.c -L../lib -lcm_api
[root@localhost modules]#
```

Els arxius executables es trobaran a la carpeta /bin dins el directori /modules.

El següent pas és fer córrer els CM daemons, d'altra manera els mòduls no funcionaran. Per això, cal obrir quatre terminals (un per mòdul) i executar respectivament en cada una d'elles des del directori /bin que penja directament del CM les següents comandes:

Terminal 1
[root@localhost bin]# ./runtbd –p 8201

Terminal 2
[root@localhost bin]# ./runtbd –p 8202

Terminal 3
[root@localhost bin]# ./runtbd –p 8203

Terminal 4
[root@localhost bin]# ./runtbd –p 8204

Exemple Terminal 3

```
[root@localhost ~]# cd ..
[root@localhost /]# cd ..
[root@localhost /]# cd root
[root@localhost ~]# cd Desktop
[root@localhost Desktop]# cd projecte20
[root@localhost projecte20]# cd bin
[root@localhost bin]# ./runtbd -p 8203
```

Una vegada fet això, cal executar des d'altra terminal la Shell, una interfície basada en comandes per controlar l'aplicació:

Terminal 5
[root@localhost bin]# ./shell –f ./shell_files/shell.conf

Execució de la Shell

```
[root@localhost bin]# ./shell -f ./shell_files/shell.conf
Connecting with 4 ARROWS machines. Please wait...
Testbed Machine Mask: 0x0000001e
=========================
Connecting with Alpha.
Main connection ready with Alpha.
Secondary connection ready with Alpha.
=========================
Connecting with Beta.
Main connection ready with Beta.
Secondary connection ready with Beta.
=========================
Connecting with Gamma.
Main connection ready with Gamma.
Secondary connection ready with Gamma.
=========================
Connecting with Zeta.
Main connection ready with Zeta.
Secondary connection ready with Zeta.
=========================
IP information transfered.
Services up in <Alpha>.
Services up in <Beta>.
Services up in <Gamma>.
Services up in <Zeta>.
RUNTB identification complete (NO error)
All clients are allowed to use services (NO error)
Initial time transfered (NO error)
Requestd modules (if any) started (NO error)
Connection with all (4) satellite machines successful

ARROWS Testbed Command Line Interpreter
Type 'help' to get a list of valid commands.
>>
```

Si tot ha anat bé, els serveis quedaran ‘up’ sense errors.

En aquest moment ha quedat inicialitzada l’eina Communications Manager. La finestra on s’ha executat la Shell espera instruccions.

Ara és el moment d’executar els mòduls, per això cal obrir quatre noves terminals (una per màquina real o emulada) i introduir:

Terminal 1
[root@localhost bin]# ./mod101

Terminal 2
[root@localhost bin]# ./mod102

Terminal 3
[root@localhost bin]# ./mod103

Terminal 4
[root@localhost bin]# ./mod104

```
[root@localhost ~]# cd ..
[root@localhost /]# cd root
[root@localhost ~]# cd Desktop
[root@localhost Desktop]# cd projecte20
[root@localhost projecte20]# cd modules
[root@localhost modules]# cd bin
[root@localhost bin]# ./mod101
```

Els mòduls ara han quedat enregistrats. Es pot veure el detall del procés de l'operació a les finestres on s'han executat els corresponents RUNTBD i la SHELL.

A.2. Execució de l'aplicació

Si tot el procés de la secció anterior ha concluit correctament, és el moment de fer córrer l'aplicació. Per això, anem a la terminal on està oberta la SHELL i inicialitzem l'aplicació "MyWlan":

>> init

Els quatre mòduls creen els fluxos de comunicacions corresponents entre ells. Els mòduls queden doncs inicialitzats.

Una vegada fet això cal introduir la següent comanda:

>> run

És just en aquest moment quan el cor de l'aplicació comença a funcionar. Les terminals on s'han executat els mòduls van donant informació del procés.

Durant l'execució és possible utilitzar altres comandes per controlar els mòduls. Aquests permeten temporalment aturar l'execució o debugar pas a pas. Alguns d'aquestes comandes són:

>> pause

>> step

>> step 10

Finalment per finalitzar l'execució utilitzem l'ordre:

>> stop

Per tal de tornar a executar tot l'operatiu sense haver de fer totes les crides fins ara explicades, es pot utilitzar la comanda:

>> restart

Aquesta reinicialitzarà l'estat del CM i es començarà una nova execució dels mòduls.

Quan definitivament es vulgui sortir utilitzarem:

>> exit

```
Requestd modules (if any) started (NO error)
Connection with all (4) satellite machines successful

ARROWS Testbed Command Line Interpreter
Type 'help' to get a list of valid commands.
>>init
Setting testbed STATUS to INIT
Command completed successfuly in <Alpha>
Command completed successfuly in <Beta>
Command completed successfuly in <Gamma>
Command completed successfuly in <Zeta>
>>run
Setting testbed STATUS to RUN
Command completed successfuly in <Alpha>
Command completed successfuly in <Beta>
Command completed successfuly in <Gamma>
Command completed successfuly in <Zeta>
>>stop
Setting testbed STATUS to STOP
Command completed successfuly in <Alpha>
Command completed successfuly in <Beta>
Command completed successfuly in <Gamma>
Command completed successfuly in <Zeta>
>>
```

A.3 Exemple d'enviament i recepció d'un paquet.

L'objectiu d'aquesta secció és mostrar els resultats que produeixen les principals funcions dels quatre mòduls per tal de verificar el correcte funcionament de l'aplicació "MyWlan".

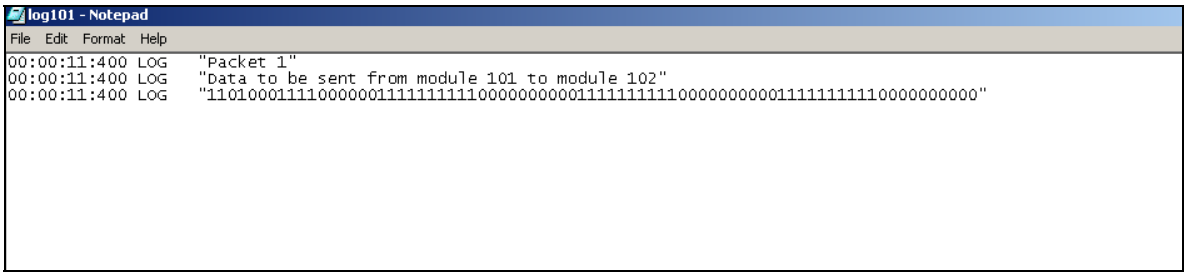
Per tal de facilitar el seguiment del programa, s'ha optat per incloure a cada subapartat algunes captures del programa funcionant en temps real a més d'una còpia dels logs generats pel CM.

Mòdul 101, Capa MAC

Aquest mòdul genera cíclicament un paquet de testeig per tal que la resta de mòduls el processin. Aquest paquet simula el que generaria la capa immediatament superior a la física, la capa MAC. Les dades de prova, un total de 77 bits es descomposen de la següent manera:

RATE 1101 (6 Mbps, BPSK)
RESERVED 0
LENGTH 001111000000 (60_d)
PSDU
11111111110000000000111111111100000000001111111110000000000

L'objectiu doncs és que aquestes dades residents al mòdul 101(capa MAC) passin mitjançant el CM al mòdul 102 (capa PLCP).
A continuació es presenten dues imatges, la primera és el log que es genera en enviar el paquet.




```
Temps d'enviament: 90 microsegons.
Sent OK

Temps d'enviament: 86 microsegons.
Sent OK

Temps d'enviament: 89 microsegons.
Sent OK

Temps d'enviament: 94 microsegons.
[root@localhost bin]#
```

Mòdul 102, 802.11 TX

Una vegada aquest mòdul ha rebut les dades de l'anterior (capa MAC) mitjançant les funcions específiques del CM es comença a operar amb els bits fins arribar al resultat final a passar a la capa PMD.

L'esquema que segueix el programa i per tant els resultats que es presenten en aquesta secció correspon a l'explicat en l'apartat 4.4.

D'acord a les dades rebudes del mòdul anterior (Capa MAC) es sap que:

1. La velocitat de transmissió del camp DATA serà de 6 Mbps.
2. Cada símbol DATA contindrà 24 bits de dades útils i 48 després de fer la convolució.
3. S'enviaran quatre símbols DATA ja que la suma de longituds del PSDU, bits de Tail, SERVICE i bits de Padding fan 96 bits (24 bits per símbol).

Ja que l'arxiu .log que genera CM automàticament presenta les dades molt atapeïdes, es presenta a continuació una còpia dels resultats en un format més intel·ligible.

SÍMBOL SIGNAL

msg.signal: 110100011110000001000000

bits_encoded: 111010111001100000101001101001011111011111001011

bits_interleaved: 101101111001110001101111111001010100101010001011

bit_signal: 0,0,0,0,0,0,0,0,0,0,0,0,1,0,-1,0,1,0,1,0,-1,0,-1,0,1,0,1,0,1,0,1,0,-1,0,-1,0,1,0,1,0,1,0,1,0,-1,0,-1,0,-1,0,1,0,-1,0,1,0,-1,0,1,0,1,0,1,0,1,0

SÍMBOLS DATA

msg.data:

000000000000000011111111110000000000111111111100000000001111111110000000000000000000000000

bits_scrambled:

000011101111001000110110110000100010100111010010101101101111001100010100111001111011010000101010

data_convol:

000000001110010111110011101001001000110001110100101000010010101011111111111011100111101100100111010010011010111000101011111010011111111110100111011100111100000101011100000000000000000000000000000000

data_interleaved:

000000001110010111110011101001001000110001110100101000010010101011111111111011100111101100100111010010011010111000101011111010011111111110100111011100111100000101011100000000000000000000000000000000

bit_data:

0,0,0,0,0,0,0,0,0,0,0,0,-1,0,1,0,1,0,-1,0,1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,1,0,-1,0,-1,0,1,0,1,0,-1,0,1,0,-1,0,-1,0,1,0,-1,0,0,0,1,0,1,0,-1,0,1,0,-1,0,1,0,0,0,1,0,1,0,1,0,-1,0,-1,0,1,0,1,0,-1,0,-1,0,-1,0,1,0,1,0,1,0,-1,0,0,0,-1,0,-1,0,1,0,-1,0,-

1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,-1,0,-1,0,1,0,1,0,1,0,1,0,1,0,1,0,-
1,0,1,0,1,0,-1,0,1,0,1,0,-1,0,1,0,-1,0,1,0,-1,0,1,0,1,0,1,0,1,0,1,0,0,0,-1,0,1,0,-1,0,-1,0,1,0,-
1,0,0,0,1,0,1,0,1,0,-1,0,-1,0,-1,0,1,0,1,0,-1,0,-1,0,1,0,1,0,1,0,0,0,1,0,1,0,-1,0,-1,0,-
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,-1,0,-1,0,1,0,1,0,-1,0,1,0,1,0,1,0,-
1,0,1,0,-1,0,1,0,1,0,-1,0,1,0,1,0,1,0,1,0,1,0,-1,0,1,0,1,0,1,0,0,0,-1,0,1,0,-1,0,1,0,-1,0,-
1,0,0,0,-1,0,1,0,1,0,1,0,-1,0,1,0,1,0,-1,0,1,0,-1,0,1,0,-1,0,-1,0,0,0,1,0,1,0,-
1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,-
1,0,-1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,-1,0,-1,0,0,0,-1,0,-
1,0,-1,0,-1,0,-1,0,-1,0,0,0,-1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,0,0,-
1,0,-1,0,1,0,-1,0,-1,0,0,0,0,0,0,0,0,0,0,0

La figura anterior està dividida en dues seccions, una primera corresponent a les dades relacionades amb el símbol SIGNAL i la següent amb el/s símbol/s DATA.
Cada secció presenta els diferents vectors (corresponents a les diferents funcions) que segueixen els bits d’acord a les especificacions.
En el cas del símbol SIGNAL seguiria el següent esquema:

Dades originals → dades convolucionades → dades interleaved → dades mapades

Pel que fa als símbols DATA:

Dades originals → dades scrambled → dades convolucionades → dades interleaved → dades mapades

```
 S'han rebut 77 bits de prova del mòdul 101.

Símbol SIGNAL enviat OK

Símbol de dades enviat OK

Símbol de dades enviat OK

Símbol de dades enviat OK

Símbol de dades enviat OK

Temps de procés per enviar paquet complet: 531 microsegons.
[root@localhost bin]#
```

Mòdul 103, 802.11 RX

Una vegada rebuts els símbols enviats pel mòdul 802.11 TX cal descodificar-los per recuperar les dades inicials.

A continuació una còpia dels resultats obtinguts:

SÍMBOL SIGNAL

bits_rebuts: 0,0,0,0,0,0,0,0,0,0,0,0,1,0,-1,0,1,0,1,0,-1,0,-1,0,1,0,1,0,1,0,1,0,-1,0,-1,0,1,0,1,0,1,0,-1,0,-1,0,-1,0,1,0,-1,0,1,0,-1,0,1,0,1,0,1,0,1,0

bits_interleaved: 101101111001110001101111110010101001010100010111

bits_deinterleaved: 111010111001100000101001101001011111011111001011

msg.signal (bits descodificats per Viterbi): 110100011110000001000000

SÍMBOLS DATA

data_rebuts: 0,0,0,0,0,0,0,0,0,0,0,0,-1,0,1,0,1,0,-1,0,1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,1,0,-1,0,-1,0,1,0,1,0,-1,0,1,0,-1,0,-1,0,1,0,-1,0,0,0,1,0,1,0,-1,0,1,0,-1,0,1,0,0,0,1,0,1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,1,0,1,0,1,0,-1,0,0,0,-1,0,-1,0,1,0,-1,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,-1,0,-1,0,1,0,1,0,1,0,1,0,1,0,1,0,-1,0,1,0,1,0,-1,0,1,0,1,0,-1,0,1,0,-1,0,1,0,-1,0,1,0,1,0,1,0,1,0,1,0,0,0,-1,0,1,0,-1,0,-1,0,1,0,-1,0,0,0,1,0,1,0,1,0,-1,0,-1,0,-1,0,1,0,1,0,-1,0,-1,0,1,0,1,0,1,0,0,0,1,0,1,0,-1,0,-1,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,-1,0,-1,0,1,0,1,0,-1,0,1,0,1,0,1,0,-1,0,1,0,-1,0,1,0,1,0,-1,0,1,0,1,0,1,0,1,0,1,0,-1,0,1,0,1,0,1,0,0,0,-1,0,1,0,-1,0,1,0,-1,0,-1,0,0,0,-1,0,1,0,1,0,1,0,-1,0,1,0,1,0,-1,0,1,0,-1,0,1,0,-1,0,-1,0,0,0,1,0,1,0,-1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,-1,0,-1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,-1,0,-1,0,0,0,-1,0,-1,0,-1,0,-1,0,-1,0,-1,0,0,0,-1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,-1,0,-1,0,1,0,-1,0,-1,0,-1,0,0,0,-1,0,-1,0,1,0,-1,0,-1,0,0,0,0,0,0,0,0,0,0,0,0

data_interl:
0000000011100101111100111010010010001100011101001010000100101010111111111
1101110011110110010011010010011010110001010111110100111111111010011101110
0111100001010111000000000000000000000000000000

data_deinterl:
0000000011100101111100111010010010001100011101001010000100101010111111111
1101110011110110010011010010011010110001010111110100111111111010011101110
0111100001010111000000000000000000000000000000

data_deconvol:
0000111011110010001101101100001000101001110100101011011011110011000101001
110011110110100001010

msg.data (bits_descrambled):

000000000000000011111111100000000001111111110000000000111111111000000000000000000000000000

En el cas de recepció, l'esquema que es segueix pel símbol SIGNAL és el següent:

Dades rebudes → dades deinterleaved → dades descodificades (Viterbi) → dades originalment enviades (si es donen les condicions adequades).

Pel que fa als símbols DATA:

Dades rebudes → dades deinterleaved → dades descodificades (Viterbi) → dades descrambled → dades originalment enviades (si es donen les condicions adequades).

Finalment el resultat obtingut passa al mòdul 104 (PSDU) que presenta les dades en un format més compacte.

```
recepció
 ok
lectura signal 1

El numero de simbols es 4

Temps de procés per descodificar paquet complet: 5168 microsegons.
[root@localhost bin]#
```

Mòdul 104, PSDU

La funció d'aquest mòdul no és més que la de guardar les dades que s'han anat rebent en un format més clar, arribant a obtenir en última instància el PSDU, que seran les dades realment útils per capes superiors.

data_result:

000000000000000011111111100000000001111111110000000000111111111000000000000000000000000000

PSDU: 111111111000000000011111111100000000001111111110000000000

```
Temps de recepció: 6 microsegons.
[root@localhost bin]#
```

A.4. Resultats

L'apartat anterior, A.3, és un exemple real del funcionament del programa. Com es pot observar, les dades enviades originalment i les rebudes a l'altre extrem coincideixen. Això vol dir que la totalitat del procés s'ha realitzat correctament.

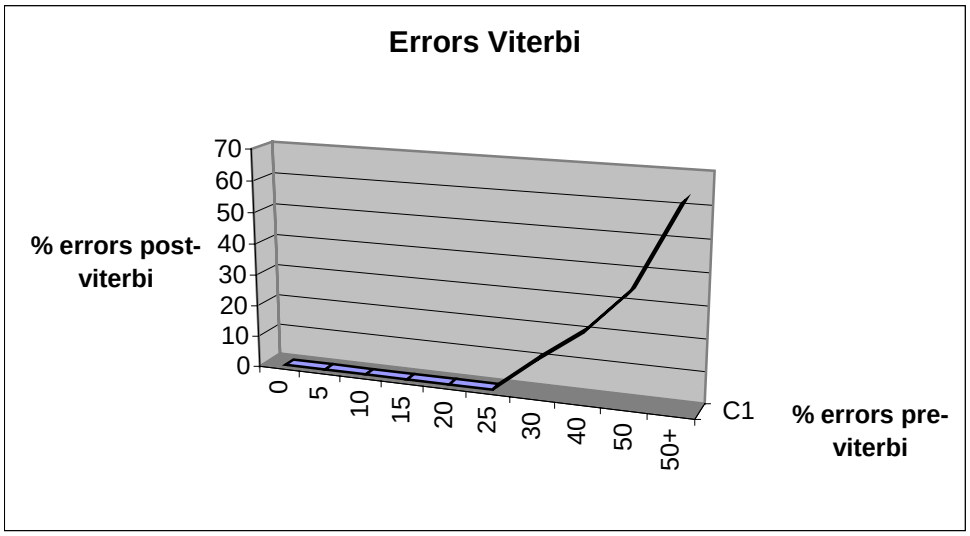
Degut a que en aquest cas és una simulació i el canal és fictici i perfecte, no s'han comès errors, però en un escenari real això no hagués passat, s'haguessin generat múltiples errors per causa del soroll.

El codi programat disposa d'un codificador convolucional en emissió i un descodificador de Viterbi en recepció que fan robust el sistema en els casos de pèrdua d'informació.

Ja que el canal per sí sol no genera errors i amb l'objectiu d'ampliar el banc de proves, s'ha forçat que alguns bits fossin erronis de manera aleatòria.
L'objectiu d'aquest test és veure quants bits cal que arribin incorrectament al receptor per tal que aquest no aconsegueixi descodificar els bits correctament.

Cal tenir en compte que hi ha configuracions a les quals se'ls aplica puncturing i per tant, s'estan eliminant bits útils, que més tard seran substituïts per 0's. Evidentment el receptor contempla com errors els 0's insertats que realment eren 1's.

A continuació es mostra una gràfica que relaciona el percentatge de bits que arriben erròniament al receptor i el percentatge de bits que el Viterbi descodifica incorrectament. (Les proves han estat realitzades per un únic paquet de 512 bits per una modulació BPSK, sense interleaving. D'aquí que puntualment es superi el 50% de bits rebuts erronis.).



Com es pot veure, el descodificador de Viterbi fa una funció important recuperant la majoria de bits enviats originalment quan el percentatge d'errors rebuts abans del Viterbi no superen aproximadament el 25%.

Un dels dubtes que van sorgir és que com era possible que l'interleaving modifiqués un 33% dels bits i en canvi el descodificador de Viterbi recuperés la totalitat de bits correctament si segons la gràfica apròximadament en arribar al 25% es comencen a generar errors. Això es deu a que no tots els bits insertats a 0's per la funció de deinterleaving van ser errors. Si la probabilitat d'enviament de 1's i 0's és la mateixa, el 50% dels bits modificats a 0's serien originalment 1's i la resta 0's, per tant, el Viterbi està corregint realment un 15-16% d'errors reals.

Pel que fa al temps de processament es tenen els següents resultats per mòdul:

Mòdul 101: Capa MAC

```
Temps d'enviament: 95 microsegons.
```

Mòdul 102: 802.11 TX

```
Temps de procés per enviar paquet complet: 499 microsegons.
```

Mòdul 103: 802.11 RX

```
Temps de procés per descodificar paquet complet: 5579 microsegons.
```

Mòdul 104: PSDU

```
Temps de recepció: 17 microsegons.
```

Amb aquests valors podem dir que per enviar un paquet com el de l'exemple, format per un símbol SIGNAL i 4 símbols DATA (un total de 128 chars i 128*4 enters, per tant, 5120 bits) calen de l'ordre de 600 µs i per rebre aquest paquet PPDU calen aproximadament 5.600 µs.

Les especificacions diuen que s'ha de poder arribar a enviar i rebre informació a 54 Mbps, això vol dir que el paquet d'exemple s'havia d'haver pogut enviar i rebre en un temps menor a 100µs per TX i altres 100µs per recepció. Estem una mica lluny d'aquests resultats.

Això no és un problema del codi, sinó de l'estructura sota la que s'està executant l'aplicació. Per un costat tenim el CM que assigna CPU d'acord unes regles, per altra banda Linux que no dedica el 100% dels recursos a l'aplicació "MyWlan" i a més, un processador Intel Pentium 4 a 3.2 GHz no és una DSP, essent aquesta última molt més ràpida [10].

És evident que no té massa sentit executar l'aplicació en aquest escenari, però sí que és important haver-ho fet primerament així per poder debugar els errors que han anat apareixent, i ara sí, una vegada que tot el codi respon correctament, traslladar-ho a una DSP per tal d'aconseguir els temps establerts a les especificacions.